BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
and Communication

BACHELOR'S THESIS

Brno, 2020 Vladimír Janout

BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS
ÚSTAV TELEKOMUNIKACÍ

APPLICATION FOR ELLIPTIC CURVE CRYPTOGRAPHY
APLIKACE PRO KRYPTOGRAFII ELIPTICKÝCH KŘIVEK

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR — **Vladimír Janout**
AUTOR PRÁCE

SUPERVISOR — **M.Sc. Sara Ricci, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Vladimír Janout ***ID:*** 203706
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Aplikace pro kryptografii eliptických křivek

POKYNY PRO VYPRACOVÁNÍ:

Student nastuduje problematiku kryptografie eliptických křivek, tj. definice eliptické křivky, vlastnosti, základní EC operace, princip bilineárního párování a základních EC protokolů. Výstupem práce bude funkční grafická aplikace umožňující provádět základní operace nad eliptickou křivkou až po výpočty základních EC kryptografických protokolů. Důraz bude kladen na uživatelskou přívětivost aplikace. Součástí práce bude také uživatelský manuál k vyvinuté aplikaci.

DOPORUČENÁ LITERATURA:

[1] Washington LC., “Elliptic curves: number theory and cryptography.” CRC press; 2008 Apr 3.

[2] The Sage Reference Manual - SageMath Documentation [online]. The Sage Development Team, 2019 [cit. 2019-09-15]. Dostupné z: https://doc.sagemath.org/html/en/reference/

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 8.6.2020

Vedoucí práce: M.Sc. Sara Ricci, Ph.D.

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRACT

Cryptography based on elliptic curves is a very popular form of public-key cryptography, however, it is often viewed only as black-box without deeper knowledge of mechanisms on the elliptic curves. This bachelor's thesis aims to uncover some of these mechanisms and describes the design and solution of graphical application for elliptic curve cryptography, providing its users with a better insight on this problematic. It does so via enabling the user to perform elliptic curve operations, showing the graph of elliptic curve over real numbers and alowing for key establishment via ECDH protocol. The result is achieved due to mutual cooperation of JavaFX desktop graphical application and SageMath Server, which serves as a mathematic engine for computations on the elliptic curve.

KEYWORDS

JavaFX, SageMath, Elliptic Curves, Cryptography, Desktop application

ABSTRAKT

Kryptografie založená na eliptických křivkách je velmi populární forma asymetrické kryptografie, na kterou se však často nahlíží bez hlubší znalosti vnitřních mechanismů na eliptické křivce. Tato bakalářská práce si klade za cíl některé mechanismy osvětlit a popisuje návrh a tvorbu grafické aplikace, která umožňuje jejímu uživateli lepší vhled do problematiky. Toho dosahuje pomocí toho, že uživateli umožňuje provádět operace nad eliptickou křivkou, zobrazuje graf eliptické křivky na množině reálných čísel a poskytuje možnost modelového ustanovení tajného klíče pomocí ECDH protokolu. Výsledku je dosáhnuto díky spolupráci mezi grafickou aplikací JavaFX a SageMath serverem, který zajišťuje výpočty na eliptické křivce.

KLÍČOVÁ SLOVA

JavaFX, SageMath, Eliptické křivky, Kryptografie, Desktopová aplikace

JANOUT, Vladimír. *Application for Elliptic Curve Cryptography*. Brno, 2020, 61 p. Bachelor's Thesis. Brno University of Technology, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Advised by M.Sc. Sara Ricci, Ph.D.

ROZŠÍŘENÝ ABSTRAKT

Tato bakalářská práce se věnuje problematice kryptografie založené na eliptických křivkách. Cílem práce bylo navrhnout a implementovat grafickou aplikaci, která by jejímu uživateli umožnila provádět základní operace nad eliptickou křivkou (sčítání a násobení bodů na křivce) a ustanovení klíče pomocí Diffieho-Hellmanova protokolu s využitím eliptických křivek (Elliptic Curve Diffie-Hellman, dále jen ECDH). Výstupem této práce je JavaFX grafická aplikace a SageMath Server, který zajišťuje výpočty na eliptické křivce. Důraz měl být kladen na uživatelskou přívětivost aplikace.

Teoretická část je zaměřena zejména na popis implementované problematiky. Jsou představeny eliptické křivky a eliptické křivky definované nad konečnými tělesy. Protože jsou konečná tělesa významně spojená s teorií grup, jsou uvedeny také. Velká pozornost je věnována operacím na eliptické křivce, protože je na nich založena kryptografie na bázi elitpických křivek. Základními operacemi jsou sčítaní bodů a skalární násobení bodů, což není nic jiného, než opakované sčítání stejného bodu k sobě samému. V další části je popsán diskrétní logartimus nad eliptickou křivkou (Elliptic Curve Discrete Logarithm Problem, dále jen ECDLP) jako analogie klasického problému diskrétního logaritmu (Discrete Logarithm Problem, dále jen DLP). Jeho výhodou je, že narozdíl od DLP nejsou pro jeho řešení známé žádné subexponenciální algoritmy, nejlepší algoritmy mají plně exponenciální složitost. V důsledku toho mají bezpečné kryptosystémy konstruované nad eliptickou křivkou výrazně kratší délku klíče, což vede k nižším nárokům na paměť a současně zvyšuje rychlost v porovnání např. s kryptosystémy na bázi DLP. Na ECDLP je založen protokol ECDH. Nejdříve se práce věnuje popisu klasického DH protokolu s využitím multiplikativní grupy, poté je představena jeho varianta s využitím eliptických křivek.

Praktická část bakalářské práce se věnuje především postupu implemetace řešení a popisuje principy za jejichž pomoci bylo dosaženo výsledku. Dle zadání mohl být k implementaci použit open-source počítačový algebraický systém (CAS) SageMath. V rámci jeho analýzy bylo zjištěno, že SageMath podporuje všechny potřebné operace s eliptickými křivkami, zejména definici eliptické křivky nad koneč-ným tělesem, získání řádu křivky, vypsání bodů náležící eliptické křivce a operaci sčítání bodů. Jeho nevýhodou pro řešení této práce je to, že neposkytuje přívětivé uživatelské prostředí. To bylo vyřešeno návrhem modelu, který využívá SageMath pouze k výpočtům nad eliptickou křivkou. Tyto výpočty jsou poté uživateli zobrazeny v grafické aplikaci, založené na platformě JavaFX, která zároveň slouží jako generátor vstupů v závislosti na požadavcích uživatele. JavaFX je platforma pro vývoj desktopových grafických aplikací v jazyce Java a je koncipována jako soubor knihoven v tomto jazyce. K fukčnosti tohoto modelu muselo být zajištěno,

aby tyto dvě aplikace spolu mohli komunikovat. Toho bylo dosaženo za pomocí vlastního řešení, které využívá model klient-server. Za pomoci Python skriptu je v prostředí SageMathu spuštěn server s otevřeným socketem, který naslouchá na adrese localhosta a nerezerovaném portu 8888. JavaFX aplikace v navrhnutém modelu reprezentuje klienta. Kdykoliv je potřeba obsloužit požadavek, klientská aplikace vygeneruje zprávu na základně parametrů předaných uživatelem, zapouzdří ji, a pošle ji serveru na zpracování. Python server tuto zprávu zachytí a pošle ji do vstupu SageMath prostředí, které obslouží výpočet požadavku a vrátí výstup. Tento výstup je znovu zachycen pomocí Python skriptu, který nezformátovaný textový výstup zašle klientské JavaFX aplikaci jako odpověď na její požadavek. V rámci klientské aplikace jsou implementovány metody, které vrácená data zformátují a přehledně zobrazí uživateli v grafické podobě (např. seznam bodů na eliptické křivce je zobrazen v tabulce spolu s jejich řády). S těmito daty může uživatel dále pracovat a interaktivně sčítat různé body na křivce, nebo je násobit za pomoci výběru těchto bodů z tabulky a stisknutí tlačítka. Umožněno je také ustanovení symetrického tajného klíče za pomoci ECDH protokolu. Uživatel v navrhnutém scénaři představuje obě komunikující strany, Alici i Boba. Nejprve stanoví parametry domény pomocí volby koeficientů eliptické křivky a charakteristiky konečného tělesa. Poté zvolí výchozí bod pro obě komunikující strany, čímž dokončí volbu veřejných parametrů. Dále zvolí tajné hodnoty α i β pro Alici a Boba a s pomocí tlačítka zajistí výměnu vypočtených bodů, což postupně vede až k ustanovení společného tajného klíče. V tomto řešení uživatel pracuje se zkrácenou Weierstrassovou formou eliptické křivky a konečným tělesem $\mathbb{F}_p$. Důvod volby této formy a tělesa je fakt, že se na této kombinaci nejlépe předvádějí mechanismy výpočtů nad eliptickými křivkami.

Součástí uživatelské přívětivosti aplikace je i snadná instalace a následné spuštění, což může být problematické u komplexnějších Java projektů, které využívají rozšiřující knihovny. Z tohoto důvodu byla aplikace zkompilována pomocí nástroje Ant a byl vytvořen instalátor, pomocí kterého je aplikace dodávána a značně zjednodušuje oba aspekty. Tento instalátor v sobě obsahuje veškeré závislosti, které jsou nutné ke spuštění JavaFX aplikace (knihovny JavaFX, Java runtime environment, rozšiřující zdroje projektu, obrázky, atd.). Jako další požadavek musí mít uživatel nainstalovaný SageMath ve verzi 8.9 na svém zařízení a zkopírovat Python skript, který má na starosti spuštění SageMath serveru do domovského adresáře SageMathu. Spouštění projektu by pak mělo probíhat pouze pomocí JavaFX aplikace, která v sobě obsahuje metodu na automatické spuštění serveru. Kompletní popis instalace a spuštění je popsán v příloze C.

DECLARATION

I declare that I have written the Bachelor's Thesis titled "Application for Elliptic Curve Cryptography" independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the thesis and listed in the comprehensive bibliography at the end of the thesis.

As the author I furthermore declare that, with respect to the creation of this Bachelor's Thesis, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll., Section 2, Head VI, Part 4.

Brno ....

author's signature

ACKNOWLEDGEMENT

I would like to sincerely thank my supervisor M.Sc. Sara Ricci, Ph.D. for her continuous guidance, motivation, and faith in me, even when I was losing trust in myself. Her comments and deep insight into the subject has greatly helped me to fulfill the task of this thesis. I would like to also express gratitude to my friends and family. Without their support, finishing this work would not be possible.

Contents

List of Figures

List of Tables

Listings

Introduction

In today's world driven by technology, we could hardly imagine our life without cryptography, as it is one of the important tools to ensure secure transfer of information. In fact everybody, who owns a smart phone or has ever signed into internet banking, came into touch with cryptography, without even realising, because it is running in the background. Since the Internet is a public place and is not secure on its own, the motivation for securing communication is substantial. Secure communication can be ensured by, for example, using fast symmetric ciphers, which have the same key for encryption and decryption. However there arises a problem with secure distribution of this key, which asymmetric, or in other words, public-key cryptosystems can solve by using pair of keys, public and private. The biggest downside of assymetric cryptosystems is that due to large key sizes, they are significantly slower than their symmetric counter parts.

Using elliptic curves when designing a public-key cryptosystem can help this issue as they provide equivalent level of security, when compared to traditional public-key cryptosystems, while using meaningly shorter keys. For example, the *National Institute of Standards and Technology* (NIST) estimates in [3], that 256 bit key in an elliptic curve system provides equivalent level of security as 3072 bit key of *Rivest-Shamir-Adleman* (RSA). As development progresses, the secure value of key size rises because of increasing computational power, which can be used to break those keys. Here the elliptic curves also have the advantage as they scale better with each key size increase. To ensure equivalence of 512 bit Elliptic Curve key, we would need an overwhelming 15360 bit key for RSA.

The reason for how the *Elliptic curve cryptography* (ECC) achieve this, is that the mathematical problem that exists on them is harder to solve than problems of other assymetric cryptosystems.

The operations on the curve are quite complex, which makes the principals of the elliptic curve cryptography more challenging to understand. This brings a problem when implementing ECC schemes, as they are sometimes used only as a black box, without deeper knowledge of their mathematical properties. Because of the complexity of the security side, there are many standards that govern and define the recommended properties of ECC. Such standards include the American National Standards Institute's (ANSI) X9.62 [20], NIST's Federal Information Processing Standards publication 186-4 [18], NUMS [21], Brainpool [22], or SECG [19].

The task of this thesis is to give description of the basic problematics of ECC and develop an application which would allow the user to perform operations over elliptic curves while giving emphasis on the application to be user friendly. Namely it would plot graph of an elliptic curve over the set of real numbers, allow the user

to perform point addition and scalar point multiplication on the elliptic curve. At last, it would let the user establish a secret key using the ECDH protocol.

In the **Background** chapter we introduce the elliptic curves and elliptic curves over finite field. Because finite fields are closely connected to groups, we describe them as well. The main focus is given on operations on the elliptic curves, because they are what ECC is based upon. The basic operations are point addition and scalar point multiplication which are described in section 1.1.2. We namely describe three situations that can occur when adding points on the curve and we define the addition using geometric and algebraic representation of the curve. With scalar multiplication we define some important properties such as order of point,order of curve, and the cofactor which are used as domain parameters in ECC. We briefly mention popular forms of elliptic curves that are used, the short Weierstrass form, Barreto-Naehrig curve and Edwards curve. We then proceed to discuss the Elliptic Curve Discrete Logarithm Problem as cryptography based on elliptic curves relies on difficulty of solving this problem. We introduce the Diffie-Hellman key agreement protocol and its variant using elliptic curves.

In the **Implementation** chapter we describe the process of creating the main solution of the thesis assignment - the graphical application which allows the user to perform operations on the elliptic curve and set up the Elliptic Curve Diffie-Hellman protocol. We propose a solution that involves SageMath and JavaFX. The implementation part is divided into 3 main sections. In the first section, we describe the role of SageMath as the "engine" of computations on the curve. We give an explanation of what SageMath is and what it is capable of in terms of operations on the elliptic curve with sample code presented in Listing 2.1. The next section explains the creation of the SageMath server and Java client as a way of data exchange between the two applications. The third section introduces the JavaFX graphical desktop application. We outline what JavaFX offers and we present the final structure of the project. We give a description to all the classes we are using with some pictures showing the graphical user interface of the application. We are looking at the application both from the user point of view as well as from the programmer's point of view and giving the explanation of what features the application offers and how to navigate through it. We show code of the most important parts of the implementation and add some more to be inspected in the appendix of this thesis. Finally, we discuss the results and unveil caveats that follow from the way of implementation in this project finishing with list of further improvements, which went beyond the assignment of this thesis.

1 Background

In this chapter, we will lay some of the mathematical properties that elliptic curves offer. At first, we will define the elliptic curve along with some following rules, then we will proceed with defining the elliptic curve over a finite field. We will briefly describe the two most used finite fields in ECC. The main focus will be put on describing the operations on the curves, which are defined over a prime finite field: **point addition** and **scalar point multiplication**. We will define it first geometrically because it is easier to imagine it from a plotted graph than from an equation. Afterwards, we will proceed to the algebraic definition, which comes straight out of the geometric definition. Then, we will move to shortly introduce three forms of the elliptic curves which are used frequently in ECC, including the form we will be using in our Implementation – the Short Weierstrass form. Following with the description of Elliptic Curve Discrete Logarithm Problem as the main reason, why cryptography based on elliptic curves is working. Finally we will present cryptographic protocol based on this problem, which is frequently used in bilions of devices – Elliptic Curve Diffie-Hellman.

1.1 Elliptic Curves

An elliptic curve E is a set of points (x, y) satisfying the equation:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{1.1}$$

where a_1, a_2, a_3, a_4, a_6 are elements of the field $\mathbb{K}$. In the rest of the thesis we will assume that the curve is defined over $\mathbb{K}$. Some examples of fields that the curve can be defined over are: real numbers $\mathbb{R}$. complex numbers $\mathbb{C}$, rational numbers $\mathbb{Q}$ and finite field of q points $\mathbb{F}_q$. Curve defined by this equation is known as **generalized Weierstrass form** of the elliptic curve [4]. Any form of elliptic curve can be expressed using this form [6].

We also define 1 condition and a point $\mathcal{O}$ called the point at infinity. The conditions is

- $A, B \in \mathbb{K}$ must satisfy:

$$4A^3 + 27B^2 \neq 0. \tag{1.2}$$

The condition ensures that the curve will be smooth and there are no points on the curve where E has more than just 1 tangent line [2]. Thus, there are no singular points on the curve that would otherwise interfere when performing various operations on the curve.

The importance of defining the point $\mathcal{O}$ together with the curve might not be visible from this definition, however we will clarify that later in the thesis, namely in section 1.1.2.

Representing the points as (x, y) is known as affine coordinates.

1.1.1 Elliptic Curves over Finite Field

When designing a public-key cryptosystem, key size is usually an important specification that we want to choose based on our security and speed requirements. We define elliptic curves over finite fields to be able to set static key size and, because of their properties, finite fields also enable us to design cryptographic algorithms like *Diffie-Hellman* (DH) key exchange and *Digital signature algorithm* (DSA) to work with elliptic curves instead of modular exponentiation.

Because finite fields with their binary operations $+$ and $*$ form a **commutative group** it is appropriate to define groups first.

Groups

Remark. The binary operation $\circ$ on set M is a morphism $M \times M \longrightarrow M$.

In other words the operation $\circ$ is a rule, where every ordered pair from set M is assigned an element again from set M.

Definition 1.1.1. Let G be a set. G together with operation $\circ$ form a group $(G,\ \circ)$, which satisfies following axioms:

Tab. 1.1: Axioms that are satisfied by Groups

1.	Existence of inverse element: $\forall a \in G, \exists i \in G, a \circ i = i \circ a = e$
2.	Existence of identity element: $\exists e \in G, \forall a \in G, a \circ e = e \circ a = a$
3.	Associativity: $\forall a, b, c \in G, (a \circ b) \circ c = a \circ (b \circ c)$
4.	$\forall a, b \in G, a \circ b \in G$

If we add following axiom, than the structure forms an **Abelian group** [2].

Tab. 1.2: The Abelian group Axiom

5.	Commutativity: $\forall a, b \in G, a \circ b = b \circ a$

Definition 1.1.2. Group $(G,\ \circ)$ is **cyclic** if there exists an element $a \in G$ such as, for every $b \in G$ there exists an integer i such as $b = a^i$. Element a is called the **generator** of group G.

Finite Fields

Generally, a finite field is a field, where set $\mathbb{F}$ contains finite number of elements. The number of elements $\mathbb{F}$ is called order of the finite field $\mathbb{F}$.

Definition 1.1.3. Let $\mathbb{F}$ be a set of elements on which two binary operations + and * are defined. The set $(\mathbb{F}, +, *)$ is a **field** if the following conditions are satisfied:

- $(\mathbb{F}, +)$ is a commutative group (and 0 is the identity element).
- $(\mathbb{F} \setminus 0, *)$ is a commutative group (and 1 is the identity element).
- Multiplication is distributive over addition, i.e. for any three elements a, b and $c \in \mathbb{F}$,

$$a * (b + c) = a * b + a * c \tag{1.3}$$

Remark. It can be proven that while $m \geq 1$, then the order of field must be m-power of prime p. Such fields of order p^m are called GF_{p^m} – Galois Fields after Évariste Galois. In some literature the designation "finite field" is fully interchangeable with "Galois field".

In ECC, the 2 most widely used fields are prime field $\mathbb{F}_p$ and binary field $\mathbb{F}_{2^m}$.

Prime fields

- $\mathbb{F}_p$ is a finite field, where p is a prime number and elements of the field are $\{0, 1, \ldots, p-1\}$. It is a Galois field with $m = 1$ and it's equivalent with the set of integers $\mathbb{Z}_p$ together with operations modulo p.
- $\mathbb{F}_{2^m}$ is a finite field with $p = 2, m \geq 2$. The reason why this field is valuable is that majority of cryptographic algorithms operate with integers because of their representation in bits. When we implement such algorithm, we usually want integers that fit into a m-bit word, that is to say numbers which are in range of $0, \ldots, 2^m - 1$. This field uses polynomial arithmetic for computations of basic operations.

In result we can conclude that points on the elliptic curve defined over $\mathbb{F}_p$ or $\mathbb{F}_{2^m}$ together with the operation of addition form an Abelian group, where $\mathcal{O}$ serves as the identity (neutral) element. It belongs to the set of points on E, however it is not an affine point [1].

1.1.2 Operations on the curve

Operations on the curve are the heart of cryptography based on EC. In this section, we will present point addition and an operation which is directly based upon it – scalar point multiplication, which serves for cryptographic systems as a trapdoor

function. Trapdoor functions are critical for designing a secure public key cryptosystem and they should not be mistaken with one-way functions. Trapdoor function is easy to perform in one direction, but require a secret to perform inverse calculation efficiently. Whereas there is not defined an easy way of performing inverse calculation with one-way function.

Point addition

When we add two different points P and Q, on the curve E, the result will again be a point, which belongs to the curve E. We can define it geometrically and algebraically. These calculations are given explicitly by Group Law [1].

Firstly we will demonstrate this geometrically on E over $\mathbb{R}$.

Remark. If we draw a line through an elliptic curve over $\mathbb{R}$ it intersects the curve in no more than 3 points.

1. For adding two affine points $P = (x_p, y_p)$ and $Q = (x_q, y_q)$, with $P \neq \pm Q$, we draw a line that connects them. This line will intersect the curve in a third point, denoted R. If we reflect this point along the x-axis we get a new point, which is the sum of $P + Q$. The proof for the third intersection of the curve can be found in [11].

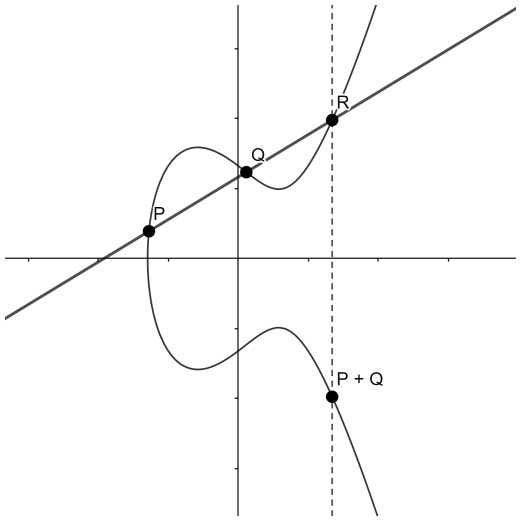

Fig. 1.1: point addition on elliptic curves

2. For adding a point P to itself, in other words doubling the point P we use similar technique as in 1. but instead of connecting line we will use a tangent line at the point P. If $y_p \neq 0$, than the tangent line will intersect the curve in a second point $-R$. The result of $P + P = R = 2P$ is a point R which is symmetric about the x axis to $-R$. If $y_p = 0$ then $P + P = 2P = \mathcal{O}$.

Remark. Points which are symmetric about the x-axis are called *inverse*.

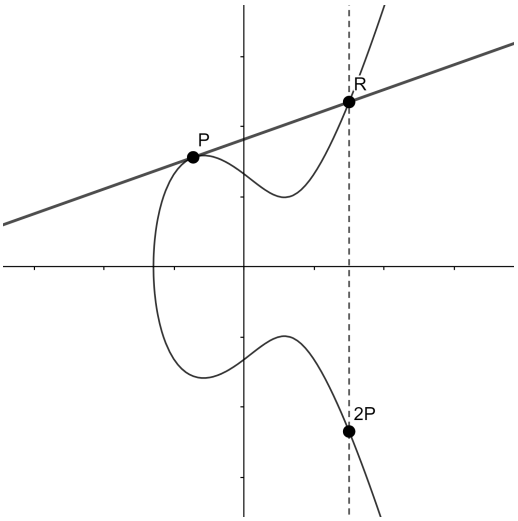

Fig. 1.2: Point doubling on E where $y_p \neq 0$

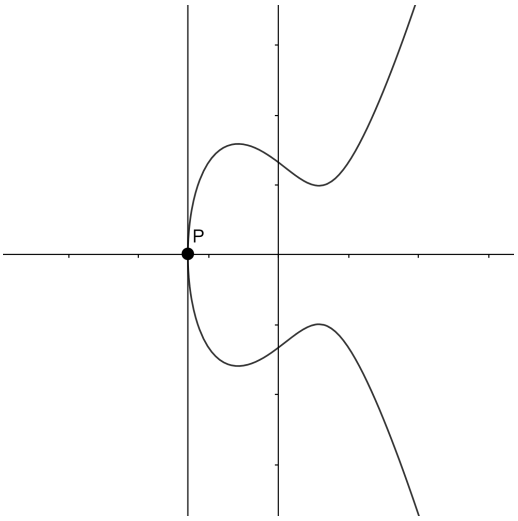

Fig. 1.3: Doubling a point on E where $y_p = 0$

3. If we sum 2 inverse points $P = (x_p, y_p)$ and $-P = (x_p, -y_p)$ then the line connecting them is parallel to the y-axis. Any line given by $x = c$ where c is constant can be imagined as passing through the point at infinity. This gives us $P + (-P) = \mathcal{O}$.

- If a line c intersects curve E in (not necessarily different) points P, Q, R then $(P + Q) + R = \mathcal{O}$.
- $P + \mathcal{O} = P$ for $\forall P, Q \in E$.

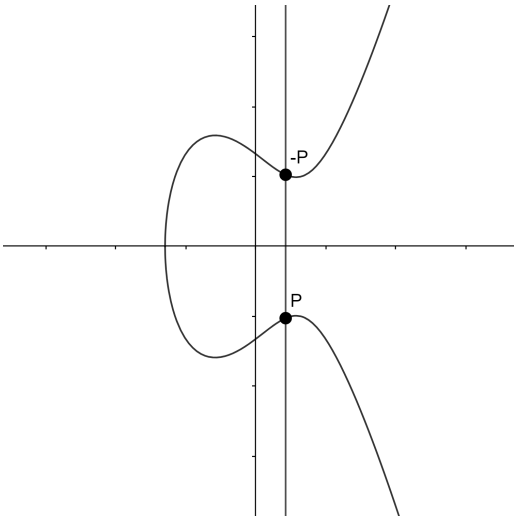

Fig. 1.4: Addition of $P + (-P)$ resulting with a point in infinity

- $P + Q = Q + P$ for $\forall P, Q \in E$.
- Let $P \in E$ then there exist a point $-P \in E$ such as $P + (-P) = \mathcal{O}$.
- Let $P, Q, R \in E$, then $(P + Q) + R = P + (Q + R)$.

Geometric interpretation of point addition should give us clear idea of how can 2 points on elliptic curve be added to receive a third point, however real systems based on elliptic curves are not normally using this representation. Because of that, we will show below the algebraic representation of point addition, which was derived from the geometric interpretation for it to be implemented using computers. We will demonstrate that on elliptic curve E in short Weierstrass form defined over finite field $\mathbb{F}_p$ because we will be using this curve and field in our implementation. The reason for choosing this particular form and field is that short Weierstrass curve is one of the most implemented forms due to the simplicity and combined with $\mathbb{F}_p$ it enables us to better explain the operations to the user in our application.

We consider a field $\mathbb{F}_p$, where $\text{char}(\mathbb{F}_p) \neq 2, 3$. Let $E : y^2 = x^3 + ax + b$ be an elliptic curve, whose points are: $\mathcal{O}$ and pairs of integers (x, y), where $x, y \in \mathbb{F}_p$ while simultaneously satisfying equation $y^2 \equiv x^3 + ax + b \pmod{p}$. Coefficients a, b are also elements of $\mathbb{F}_p$ and they meet $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$, which ensures us that following set of points form a group. Here we define:

1. $-\mathcal{O}$ as an inverse point to $\mathcal{O}$.
2. for every nonzero points $P = (x_p, y_p) \in E(F_p)$ exists $-P = (x_p, -y_p \pmod{p})$.
3. for every $P \in E(F_p)$ applies $P + (-P) = \mathcal{O}$ and $P + \mathcal{O} = P$.

Then adding two different, nonzero, not inverse points $P = (x_p, y_q)$ and $Q = (x_q, y_q)$ results in $P + Q = R$, where the coordinates of $R = (x_r, y_r)$ are calculated as follows:

$$s = (y_p - y_q)/(x_p - x_q) \pmod{p},$$
$$x_r = s^2 - x_p - x_q \pmod{p},$$
$$y_r = -y_p + s(x_p - x_r) \pmod{p}.$$

The operation of division is defined as multiplication with the inverse element, such as $x/y = x \cdot y^{-1}$, where y^{-1} is the element of $\mathbb{F}_p$ for which applies:

$$y \cdot y^{-1} \equiv 1 \pmod{p}.$$

Doubling of point P: if $y_p \neq 0$, then $2P = R$, where

$$s = (3x_p^2 + a)/(2y_p) \pmod{p},$$
$$x_r = s^2 - 2x_p \pmod{p},$$
$$y_r = -y_p + s(x_p - x_r) \pmod{p}.$$

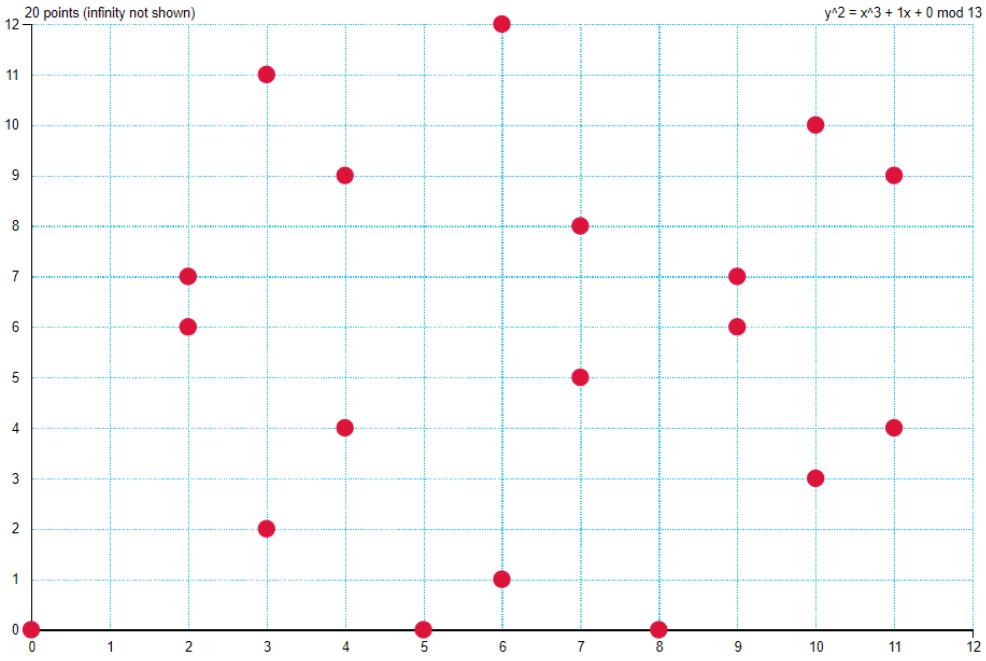

Fig. 1.5: Points of curve $E : y^2 = x^3 + x$ over $\mathbb{F}_{13}$

The procedure for addition of 2 points on the elliptic curve in Weierstrass form require 2 multiplications, 1 squaring, and 1 inversion in the field. When using an elliptic curve over finite field, real ECC systems usually use projective coordinates rather than affine coordinates, since affine coordinates are expensive over prime fields because of costly field inversions [6].

Scalar point multiplication

If we can calculate $2P = P + P$, then we can calculate $3P = (P + P) + P = 2P + P$ and we can apply this calculation for $4P, 5P, \ldots$. We define the scalar multiple n of a point P as:

$$T = n \cdot P = \underbrace{P + P + \cdots + P}_{\text{n times}} \tag{1.4}$$

If we continuously performed this operation, we would get different points xP on the curve $E(\mathbb{F}_p)$. Since an elliptic curve defined over finite field has only finitely many points and points on the curve form a cyclic group, this operation must get into loop after performing certain number (x) of steps. In the looping point xP we can state that $xP = yP$, where yP is a previous point from the field. From this statement we get:

$$xP - yP = (x - y)P = \mathcal{O}, \tag{1.5}$$

this means there exists some integer $z = x - y$,$z < x$ such as $zP = \mathcal{O}$. This means we will always get to the point in infinity after certain number of steps and then the cycle starts again.

Definition 1.1.4. Let z be a positive integer and point $P \in E$. The smallest z for which applies $zP = \mathcal{O}$ is called **order of point** P [1].

Definition 1.1.5. Order of elliptic curve E is the total count of points on the curve. It is denoted as $\#E$ [1].

Definition 1.1.6. Let P be a point with order n, with $P \in E$. The value $h = \dfrac{\#E}{n}$ is refered to as **cofactor**.

This simple method for scalar multiplication presented here is not used in practical cryptographic algorithms because of its high computational time for big numbers n. Because ECC security relies on scalar multiplication, numerous algorithms for scalar multiplication have been developed both to offer endurance against attacks and also to speed up computation. Some of the algorithms are double-and-add, add-and-double, window methods for scalar multiplication and Montgomery ladder, which is popular in cryptographic applications due to its constant time and therefore resilience against side-channel attacks [12].

Such algorithms are described in [11] along with pseudo code of given algorithms which can be used as a guide for implementation. Their cost efficiency is compared in Chapter 3.7 of [2].

1.2 Forms of elliptic curves

Elliptic curves exists in many different forms. Each of the form has its benefits and caveats and each form also has unique properties. Unfortunately we cannot present them all here. For the purpose of this thesis we will present short Weierstrass curve, because it is the is the most widespread curve. We will also present Barreto-Naehrig curve because it is pairing friendly and Edwards curve for its speed.

1.2.1 Short Weierstrass curve

Curve in short Weierstrass form [1] satisfies the following equation:

$$y^2 = x^3 + ax + b. \tag{1.6}$$

We are assuming that this is a special instance of generalized Weierstrass curve where $a_1, a_2, a_3 = 0, a_4 = a$ and $a_6 = b$, where $a, b \in \mathbb{F}_q$ [1]. This curve can be defined over $\mathbb{F}_p$ which has not characteristic equal to 2 and 3.

1.2.2 Barreto-Naehrig curve

Bareto-Naehrig curve [23] is a curve which is given by the following equation:

$$y^2 = x^3 + b, \tag{1.7}$$

where $b \neq 0$. The curve is known for having better speed than other curves.

1.2.3 Edwards curve

The Edwards curve [24] is an elliptic curve with coefficients $a_1 = dxy, a_2 = 1, a_3 = a_4 = 0$, and $a_6 = 1$ and is given by equation:

$$x^2 + y^2 = 1 + dx^2y^2, \tag{1.8}$$

where $a, d \in \mathbb{F}_{p^m} \setminus \{0\}$ with $p \neq 2$.

More forms of elliptic curves including those presented here are summarized and compared in [6].

1.3 The Elliptic Curve Discrete Logarithm Problem

In order to build cryptography based on elliptic curves, we need to find a computationally hard mathematical problem to assure security. The 2 most well-known problems nowadays that are used in modern ciphers are *Integer factorization* (IF),

used in RSA and *Discrete logarithm problem* (DLP), used in e.g. DH. We will now introduce the second named problem.

Let p be a prime and let a, b be integers that are not congruent to zero mod p. Let k be an integer such as

$$a^k \equiv b \pmod{p}.$$

The discrete logarithm problem is to find k. Since this scheme works for multiplicative group, we need to transform it for the additive abelian group on elliptic curve so that multiplication of elements $a \cdot a \cdot a \cdot \ldots$ (that is a^k) converts to point addition on the elliptic curve $P + P + P + \ldots$ (that is kP).

Let Q, R be points on elliptic curve $E_p(a, b)$ and equation $Q = kR$, where $k < p$. With the knowledge of R and k it is easy to calculate Q, but calculating k only from the knowledge of Q and P is considered computationally hard. Positive integer k is considered the discrete logarithm of point Q relative to root point R on the elliptic curve E. This problem is called *Elliptic curve discrete logarithm problem* (ECDLP).

The problem can always be attacked by brute force: trying all possible values until we find k. This becomes very impractical when k becomes very large integer, possibly several hundred digits, which is size used in real cryptographic algorithms as seen in Figure 1.6.

The ECDLP is usually considered harder problem then DLP, since currently known algorithms for solving the ECDL problem have a full exponential cost, whereas, for the classic DL problem, there are algorithms which are able to solve it in sub-exponential time. Because of that, elliptic curve cryptography can offer shorter keys while preserving same amount of security compared to non-EC cryptography [9].

However, we will introduce various special cases for which the ECDL problem can be solved by reducing it to an easier problem. Based on [10] such cases include following:

Supposing we have an elliptic curve E defined over a finite field $\mathbb{F}_p$.

1. The elliptic curve E is called **supersingular** if $\#E$[1] $= p + 1$. Problem on this particular elliptic curve can be reduced to a discrete logarithm problem on the multiplicative group. The MOV attack exploits this using the Weil pairing, details can be found in [1]. For big k it is not of a great importance, yet it shifts back the elliptic curve efficiency and enables sub-exponential algorithms to compute the DLP in these curves. The proof can be found in [5].

[1] As defined before in definition 1.1.5

2. The elliptic curve E is called **anomalous** if $\#E = p$. These curves were introduced to mitigate the application of Weil pairing which is used in the MOV attack. Unfortunately, in this case, the problem can be reduced to simple addition in $\mathbb{F}_p$ and the ECDLP can be solved in linear time, therefore even faster than in the case of supersingular curves [1].

3. Certain attacks on the DLP like the Pollard method or the Pohlig-Hellman method can be used if $\#E$ is divisible by only small primes. These methods are able to solve the DLP in $O(\sqrt{q})$ where q is the largest prime divisor of $\#E$. The methods are described in [1].

There are more methods and attacks exploiting special cases of elliptic curves and the list above is definitely not exhaustive, however, for the purpose of this thesis, it is enough to make a following point.

The conclusion from this is that curve choice and its domain parameters[2] decide how hard the problem on elliptic curves will be, which directly impacts the security of designed protocols. Poor choices could lead to degradation of the benefits that elliptic curve cryptography can offer (such as smaller key-sizes), or even make a protocol insecure.

Because testing each randomly generated curve and domain parameters would be time-consuming and hard to implement, several standardization organizations published domain parameters of elliptic curves for several common field sizes which are considered secure. These domain parameters are known as "standard curves" and can be used as a guideline when implementing protocols based on elliptic curves. Such guidelines can be found e.g. in [18], [19], [20]. To give an example, NIST recommends in [18] one elliptic curve, given in short Weierstrass form, for five prime fields $\mathbb{F}_p$ with certain prime p of sizes 192, 224, 256, 384 and 521 bits to be used as the digital signature standard for federal and government use in the US. We give an example of P-192 in the following equation and Figure which contains the recommended domain parameters.

Figure 1.6 shows the pseudo-random curve P-192, which was generated from the output of a seeded cryptographic hash function. The equation of curve is following:

$$E: \; y^2 \equiv x^3 - 3x + b \pmod{p} \tag{1.9}$$

[2]Example of domain parameters can be seen in step 1 of ECDH

With the following parameters given:

- The prime modulus p
- The prime order n
- The domain parameter seed
- The output c of the SHA-1 based algorithm
- The coefficient b
- The coordinates (G_x, G_y) of the root point.

$p =$ 6277101735386680763835789423207666416083908700390324961279

$n =$ 6277101735386680763835789423176059013767194773182842284081

$SEED =$ 3045ae6f c8422f64 ed579528 d38120ea e12196d5

$c =$ 3099d2bb bfcb2538 542dcd5f b078b6ef 5f3d6fe2 c745de65

$b =$ 64210519 e59c80e7 0fa7e9ab 72243049 feb8deec c146b9b1

$G_x =$ 188da80e b03090f6 7cbf20eb 43a18800 f4ff0afd 82ff1012

$G_y =$ 07192b95 ffc8da78 631011ed 6b24cdd5 73f977a1 1e794811

Fig. 1.6: Curve P-192. Because the order n is prime, the cofactor is always $h = 1$. Integers p and n are given in decimal form, bit strings, and field elements are given in hexadecimal.

1.4 Diffie-Hellman key agreement protocol

Key distribution was considered one of the major problems of cryptography until 1976 when the Diffie-Hellman key agreement protocol was introduced [7].

It enables its users to establish a secret symmetric key over an insecure channel without sharing any secrets beforehand or meeting each other in person. Interesting is, that by the time it was developed, its main place of use - the Internet was still in its infancy, therefore the protocol inventors had to foresee the digital revolution which happened shortly after. The protocol and its variations are the state-of-art of key exchange in cryptography. For example it is used in *Transport Layer Security* (TLS), *Secure Shell* (SSH) or the *Internet Key Exchange* (IKE) part of the *Internet Protocol Security* (IPsec) suite, so any VPN based on IPsec uses DH too. Its security lies on the difficulty of solving the discrete logarithm problem described in previous section.

It is important to note that the protocol in its presented form does not offer authentication of involved parties, therefore it is prone to *Man In the Middle* (MITM) attack. Some layer of authentication (e.g. by using digital certificates) of communicating parties must be added in order for the exchange to be secure.

There are different variations of this protocol, but for our purposes, we will mention only the basic one.

1.4.1 Diffie-Hellman using multiplicative group $\mathbb{Z}_p^*$

Supposing we have 2 parties, Alice and Bob who want to exchange data using symmetric encryption scheme such as AES. They first need to establish a secret key which will be used both for encryption and decryption.

1. Let $\mathbb{Z}_p^*$ be a finite multiplicative group and g a generator of subgroup of order q. Both of these parameters are publicly known.

2. Alice chooses a secret value $\alpha \in_R \mathbb{Z}_q$ and computes $A = g^\alpha \pmod p$ a sends A through an insecure channel to Bob.

3. Bob chooses a secret value $\beta \in_R \mathbb{Z}_q$ and computes $B = g^\beta \pmod p$ and sends B through an insecure channel to Alice.

4. Alice computes $K = B^\alpha \pmod p = (g^\beta)^\alpha \pmod p = g^{\alpha\beta} \pmod p$. She gets the same value as Bob, as he computes $K = A^\beta \pmod p = (g^\alpha)^\beta \pmod p = g^{\alpha\beta} \pmod p$. They both establish the symmetric key K.

The eavesdropper knows the group $\mathbb{Z}_p^*$, chosen generator of subgroup q and values A, B. However, he needs either secret value α or β to compute the key K. To do that, he would have to break the discrete logarithm problem. There is no known algorithm that would efficiently solve this problem in polynomial time, so for parameters $|p| \geq 1024$ bits and $|q| \geq 160$ bits the protocol is considered secure.

1.4.2 Elliptic Curve Diffie-Hellman

As we described in 1.3, there exists a particular case of the DLP on the elliptic curves, the ECDLP. Since Diffie-Hellman protocol security relies on the difficulty of solving DLP we are able to design Diffie-Hellman protocol to work with elliptic curves. The original protocol uses multiplicative group $\mathbb{Z}_p^*$, whereas the elliptic curve variant of the protocol, ECDH, [8] uses the additive abelian group which is represented by points on the elliptic curve over a finite field $\mathbb{F}_p$.

1. Let E be an elliptic curve in Short Weierstrass form defined over finite field $\mathbb{F}_p$ such that the ECDLP is hard in $E(\mathbb{F}_p)$. The curve E must be agreed upon by the communicating parties, Alice and Bob. They also agree on point $G \in E(\mathbb{F}_p)$, called the **root point** with order n. The values a, b which define the curve, together with p, G, n, h are the **domain parameters**[3], which are publicly known (usually the domain parameters are chosen, so that the order n is a large prime) [1].

2. Alice chooses private value α, such as $1 \leq \alpha \leq n-1$, computes point $A = \alpha P$ and sends A to Bob through an insecure channel.

3. Bob chooses private value β, such as $1 \leq \beta \leq n-1$, computes point $B = \beta P$ and sends B to Alice through an insecure channel.

4. Alice computes point $K = \alpha B = \alpha\beta P$ and obtains the same value as Bob, after Bob computes $K = \beta A = \beta\alpha P = \alpha\beta P$. Both parties agreed on point K. This is possible due to the commutative axiom defined in Tab. 1.2 which holds for points on our E.

5. Since in affine coordinates the point K would consist of 2 coordinates (K_x, K_y) the parties could use only the x coordinate as the symmetric key or they could use a hash-based function to derive the key from it.

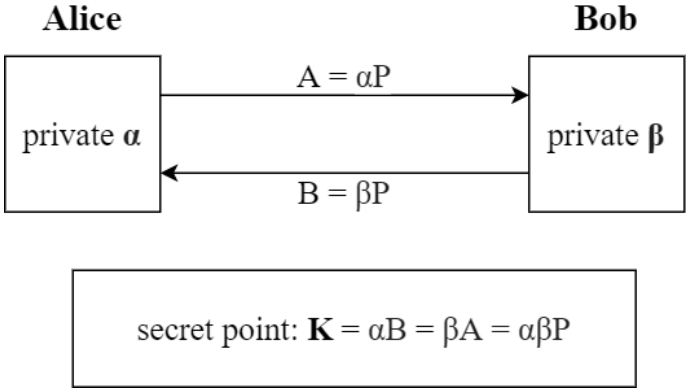

Fig. 1.7: Elliptic Curve Diffie-Hellman

[3] h is cofactor, as defined in definition 1.1.6

2 Implementation

The goal of this thesis is to create a graphical application that would allow the user to perform basic operations over elliptic curves and also allow performing calculations of an EC protocol, ECDH.

The application can serve as an educational tool and is capable of performing operations on elliptic curves described in the Background chapter, while trying to help the user understand what is happening in each step of the elliptic curve operations and elliptic curve based protocol ECDH.

According to the proposed task of the thesis, the author should have put emphasis on making the application user friendly. Therefore, console application with text input and output would not be sufficient for the fulfilment of this task.

Sage, which was recommended for the implementation, offers console input and output, which is not so easy to use for a regular user, on the other hand it is a powerful tool, that is capable of computing many complex operations which are required for calculations on the elliptic curve.

Remark. It is important to note that in our work, we are using SageMath for Windows and the graphical application is currently fully functional on machines with the Windows operating system. We gave priority to deploy on this OS as majority of personal computers are running it. However that does not mean deployment is not possible to other operating systems. There is very little platform-specific code which is mostly regarded to launching processes and navigating through OS file system, which can be fixed by running few commands in the terminal on other operating systems. The used languages, Python and Java as well as SageMath can run on Linux, or OS X based machine.

Sage is used for internal computations and the challenge was to create program with a *graphical user interface* (GUI) that would, based on the user behaviour:

- generate input for Sage
- let Sage process the calculation
- collect the output given by Sage
- format the output to a graphical layout

To give an analogy to the reader, we can imagine this project being structurally somewhat similar to developing web applications, which consists of back-end and front-end parts. In our case, Sage ensures the back-end functionalities and the implemented application serves as front-end application, which the user is interacting with.

Proposed functionalities

To be more specific of what the application should be capable of, the software should serve the user as a calculator of elliptic curve operations, which would otherwise be time consuming if done by hand and also a graphical viewer of geometrical representation of the elliptic curves. The representation is plotted in $\mathbb{R}$ as it is easier and prettier for the user than showing just points which belong to group $E(\mathbb{F}_p)$ as it is visible in Figure 1.5.

The main functionalities contain:

- plotting the elliptic curve in $\mathbb{R}$
- letting the user choose the variables of the curve, which is a, b for the curve in short Weierstrass form and the characteristics of $\mathbb{F}_p$
- displaying properties of E over $\mathbb{F}_p$ such as order of the curve, possible orders of points based on the current parameters and table of the points
- display order of points on the elliptic curves, as this is critical information for the user when choosing root point for the ECDH protocol
- perform operations on the curve such as point addition, point doubling, scalar multiplication of points
- demonstrate establishment of a symmetric key using ECDH protocol

Based on the structure we chose, it is appropriate to divide implementation part into several stages. These stages are described in following sections.

1. SageMath
2. Connection between Sage and JavaFX application
3. JavaFX application

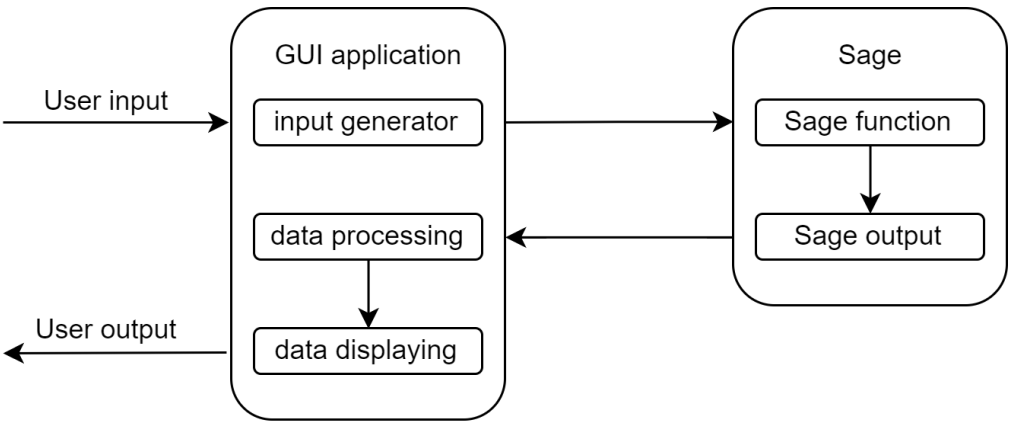

Fig. 2.1: Diagram of proposed internal logic

2.1 SageMath

SageMath is a free and open-source *Computer algebra system* (CAS) covering many aspects of mathematics. It is distributed under the GPL open source license. Sage is Python-based language and it is capable of computing various kinds of math operations such as derivatives, integrals, equations or operations on elliptic curves. Its goal is to create a capable free and open source alternative to products such as Maple, Matlab, WolframAlpha and Magma [13].

Because we want to launch and interact with the Windows version of Sage, we will shortly describe it, as it posts some caveats which will be presented further in this chapter.

After installing .exe package from the official website[1] we can use 3 main shortcuts for starting Sage. They will appear at users desktop as well as in the start menu, as shown in following Figure 2.2:

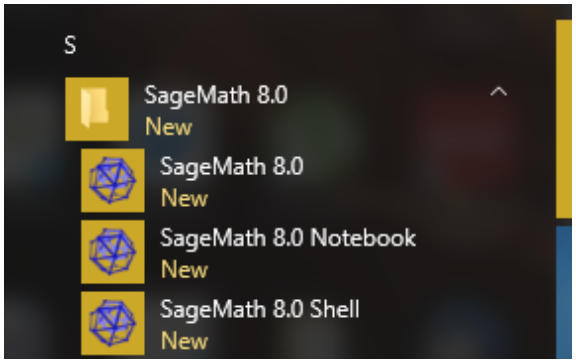

Fig. 2.2: The three options how to run SageMath in a Windows environment

The first shortcut, labeled "SageMath <version>", launches the **SageMath Console** which is a terminal emulator that takes us straight to the Sage command-line prompt. It is an equivalent of executing `sage` command in a Unix-based operating system.

Second, labeled "SageMath <version> Notebook", launches **Jupyter Notebook** server[2], with the ability to run notebooks in the Sage kernel.

The third, and the one we are interested in the most, launches the **SageMath Shell**, which is a bash shell in the Sage environment. This option enables us to execute Python scripts within Sage environment using this command:

```
sage -python (Script Path)
```

We can use `ls` to check the current working directory of Sage, which is usually by default the home directory of the current logged in user: `C:\Users\<current_user>`.

[1] `https://www.sagemath.org/download-windows.html`
[2] `https://jupyter.org`

Remark. This is true for SageMath 8.9 for Windows, with release date: 2019-09-29 and can be changed in newer versions.

Because SageMath was originally deployed for Unix-based systems, all of the shortcuts running different forms of Sage are using Cygwin Mintty emulation terminal[25], in order to use SageMath on a Windows OS machine. This brings some caveats, for example in the way the three applications are launched. We can examine that looking at full command, which is executed when the user clicks the SageMath Shell icon.

```
"C:\Users\<username>\AppData\Local\SageMath 8.9\runtime\
   bin\mintty.exe" -t 'SageMath 8.9 Shell' -i sagemath.
   ico /bin/bash --login -c '/opt/sagemath-8.9/sage -sh'
```

As we can see from the command above, in order to launch the SageMath Shell, the system needs to execute mintty.exe with certain parameters in order for the shell to start. This can be troublesome for executing processes using classes like `ProcessBuilder` in Java, as this library is normally used to launch .exe files or programs within the windows command line as can be seen in some ProcessBuilder usecases[26], where in one of the examples, the class is utilized so that Java program executes ping command and prints the output to Java console.

In the following section, we will explain functionalities SageMath can offer, which are important for our project, these include:

- Defining elliptic curve
- Order of the curve
- Coordinates of points on the elliptic curve
- Order of a point and getting a point of maximum order
- Sum of two points

2.1.1 Defining elliptic curve

There are different ways to input elliptic curve to Sage based on which form and field we are using, see [14] for an extensive description. First example, shown in Listing 2.1 on line 3 is an elliptic curve in generalized Weierstrass form $y^2+xy+3y = x^3+2x^2+4x$ over $\mathbb{F}_5$. The arguments for this command are in the same order as we defined in Equation 1.1. The last argument would not be displayed as all of our operations are performed modulus 5, so there is no need to display: $5 \mod 5 \equiv 0$.

Representing the elliptic curve in short Weierstrass form could be done by assigning 0 to the arguments which are not defined for this form, as shown on line

5, however, trying to input such a curve as in the example will result in an *Arithmetic Error*, which is visible on line 20, because this curve does not satisfy condition 1.1. We will prove this in following equation supposing we have a curve $E : y^2 = x^3 + 3x + 4$ over $\mathbb{F}_5$

$$4A^3 + 27B^2 \not\equiv 0 \mod 5$$

$$4 * (3)^3 + 27 * (4)^2 \not\equiv 0 \mod 5$$

$$400 \not\equiv 0 \mod 5$$

$$0 \equiv 0 \mod 5$$

The other way to define curve in short Weierstrass form is to specify the field in advance, and then define the curve as on line 7 to avoid the zeros and directly specify the desired form.

2.1.2 Order of the curve

Sage can return the order of a curve simply by calling a `cardinality()` function, assuming we have defined it previously and it is stored in a specified variable. Getting this variable will reveal us the number of points that elliptic curve has, which is crucial for judging the difficulty of discrete logarithm problem on the curve [1]. Concluding this, the more points E has, the bigger possibilities to make the problem harder to solve and the greater is the security.

Remark. Getting the count of points was not efficiently solved problem until 1985, as there only existed algorithms with exponential running time. In 1985 this problem was solved when Schoof's algorithm, which can count the points on the curve in polynomial time, was introduced [1].

2.1.3 Coordinates of points on the elliptic curve

With a predefined elliptic curve, Sage can output the coordinates of every point when the `points()` function is called. The outputted points as shown in 2.1 are represented by projective coordinates and we can divide them into two classes.

1. points represented in format of $(x : y : 0)$ are representing the points at infinity. $(x : y : 0) \to \mathcal{O}$
2. points represented in format of $(x : y : 1)$ are affine points on the curve and can be transformed from projective space in this way: $(x : y : 1) \to (x, y)$

2.1.4 Order of a point and getting a point with maximum order

Getting order of a point can be done for a point defined on the curve as shown in the example in listing 2.1 on line 12. This point can be specified using affine coordinates. Sage will print the order when the `order()` function is called. A very useful function that the E object has is `gens()`, which returns one point of maximum order in $E(\mathbb{F}_p)$. Order of this point is equal to the order of curve E, if such a point is present on E.

2.1.5 Sum of two points

When we define two points P, Q in the Sage console, which belong to a defined elliptic curve, we can add them using `P + Q` command. In the example in listing 2.1 is presented one of ways to perform point doubling. Sage will return a point on the curve which is a result of this operation.

```
1  # Example
2  # 1. Variant
3  # EllipticCurve(GF(5),[1,2,3,4,5])
4  # 2. Short Weierstrass
5  E = EllipticCurve(GF(5),[0,0,0,3,4])
6  k.<a> = GF(5) # F_5 Field
7  E = EllipticCurve(k,[-1,1]); E
8  # EC in field k defined before
9  print 'the order of E is', E.cardinality() #order of E
10 print 'point of E(F_5):'
11 E.points() #points of E.
12 P = E(0,1) #point
13 print 'the order of P = ', P, 'is ', P.order()
14 print ' a point of maximum order is'
15 E.gens() #one point of maximum order in E,
16 #that is a point with order equal to the order of E
17 print 'the sum P +P is', P + P #sum of two points
18
19 Output:
20 ArithmeticError: invariants (0, 0, 0, 3, 4)
21 define a singular curve
22 Elliptic Curve defined by y^2 = x^3 + 4*x + 1 over
23 Finite Field of size 5
24 the order of E is 8 point of E(F_5):
25 [(0 : 1 : 0), (0 : 1 : 1), (0 : 4 : 1), (1 : 1 : 1),
```

```
26 (1 : 4 : 1), (3 : 0 : 1), (4 : 1 : 1), (4 : 4 : 1)]
27 the order of P = (0 : 1 : 1) is 8
28 a point of maximum order is ((0 : 1 : 1),)
29 the sum P +P is (4 : 1 : 1)
```

Listing 2.1: Example of commands that can be given to Sage and the displayed output

2.2 Connection between Sage and JavaFX application

Because Sage is written in Python and our application is using Java programming language, one of the main challenges was to connect these two applications as proposed in diagram 2.1. Sage can be embedded into a web page using a feature named SageCell[3], which is a cloud service where SageMath is installed on a server and the requests and responses are sent through the internet.

We want our application to work in an offline environment so our prerequisite is that the user has Sage installed on his PC.

2.2.1 ProcessBuilder class

One of the first suggested solution for connecting the applications was using methods implemented in the `ProcessBuilder` class [17]. This class, as the name hints, is used to create processes and offers many methods to work with them, e.g. create a pipe between Java program and launched processes, redirect input and output, etc.

At first, this looked like an ideal way how to connect our JavaFX application to SageMath as the idea was to create an instance of ProcessBuilder class to create a pipe and generate input using this object and redirect output given by Sage from the Mintty console to our program.

However, after several tries and extensive time spent on examining this option, it turned out it is very difficult to execute this strategy. It was probably due to the fact, that SageMath is not a simple program, as say - notepad.exe, rather it is a complex system with many layers and creating many subprocesses with its launch on Windows, which are hard to navigate through using just object-oriented approach and methods implemented in the `ProcessBuilder` class.

[3] `https://sagecell.sagemath.org/`

The author switched to finding a different way of connection, which would possibly be less complex and more straightforward from a programming point of view. The solution found will be described in the following subsection.

2.2.2 Connection using Sockets

Another suggested solution involves using sockets. This adopts the client-server model, where SageMath represents the server-side, providing mathematical functions to our JavaFX application, which is representing the client-side. The communication between the respective nodes is shown in Figure 2.3.

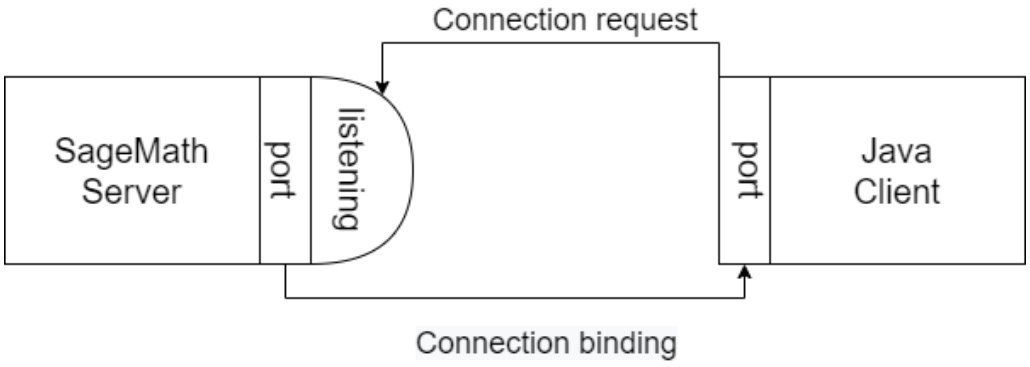

Fig. 2.3: Communication scheme of client and server using sockets

The downside of this approach, compared to using `ProcessBuilder` class is of course the necessity of building the server, which was one of the situations, where the author reached a seemingly dead end.

The Server side

The thought behind this approach was to build a server that would have access to SageMath functionalities while listening on a given port and address, so that Java client can send input, let SageMath process it and redirect the output from the SageMath console to the Java application.

The author was aware of SageMaths capability of running Python scripts within its environment and was seeking a way of writing a Python script, which would accomplish this thought.

A post [29] from user called "Woodgnome" was extremely helpful in this matter. After careful examination and testing, this solution was adopted unchanged, because it fulfills our purpose to the full extent and is very well commented.

We will provide description of this python script along with some sample code of the most important parts.

The script opens a socket on the localhost ip address `127.0.0.1` and is listening on port `8888`.

```
HOST = 'localhost'
PORT = 8888
# Create socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,
   1)
# Bind socket to localhost and port
try:
    s.bind((HOST, PORT))
# Start listening on socket
s.listen(10) #10 indicates max number of queued up
   connections before dropping requests
```

It takes the message from the connected client

```
# Receive message from client
msg = conn.recv(MAX_MSG_LENGTH)
```

And based on the contents of the message it will then either shut down or parse the message and forward it to the SageMath environment.

```
if msg == "stop":
  SHUTDOWN = True
else:
  parsed = preparse(msg)
   # Redirect stdout to my stdout to capture into a
      string
  sys.stdout = mystdout = StringIO()
```

Finally, it will let SageMath evaluate the message from the client, collect the output, forward it to the client, and close the client connection.

```
# Evalutate msg
eval(compile(parsed,'<cmdline>','exec'))
result = mystdout.getvalue() # Get result from
   mystdout
# Send response to connected client
conn.sendall(result)
```

To conclude, this allows us to generate input from our Java client and execute Sage commands in the same way, as if we were in SageMath Console. The benefit

of this approach is scalability. In our thesis we are running the server on `localhost` and we are working with a 1-to-1 approach. However if the server was deployed on a local area network or even in the Cloud, we could as well have multiple clients and provide the computation as a service, if we raised the number of queued up connections, to satisfy more clients.

The Client side

In our Java application, we implemented a method which we call `interactWithSage`. It takes a String as an argument and passes it to SageMath as the input using Socket class [27]. It then uses BufferedReader [28] to capture the output from the socket to a String variable, which it then returns to be further processed by our Java program.

```
    private String interactWithSage(String input) {
    String dataFromSage = "";
    try {
      Socket socket = new Socket("localhost", 8888 );
      OutputStream output = socket.getOutputStream();
      byte[] data = input.getBytes();
      output.write(data);
      String line;
      BufferedReader reader = new BufferedReader(new
         InputStreamReader(socket.getInputStream()));
          while ((line = reader.readLine()) != null) {
            dataFromSage += line + "\n";
          }
          socket.close();
          return dataFromSage;
    } catch (UnknownHostException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
    return dataFromSage;
  }
```

2.3 JavaFX application

JavaFX is a platform for creating user friendly desktop and internet applications. It offers a great amount of graphical and multimedia interfaces. Because its library is written as a Java *Application programming interface* (API), JavaFX code can reference APIs from any Java library (if imported) and any internal logic of the application is written in Java as well.

One of its biggest advantages is the separation of each individual component of the application. It has a lot of features and use cases, which can be found in [15], we will highlight only some of the features which can be found in this source because we only use some in our application:

1. GUI - JavaFX uses **FXML** and **Scene Builder**. FXML is an XML-based declarative markup language which is used to construct JavaFX GUI. It has a tree-structure as well as the Scene Graph, which is also object-oriented. Its elements are called nodes and each node has just one parent and optionally can have child nodes. Scene Builder serves as a FXML code generator and helps the user to interactively design the layout and also prepare functions that each node should be capable of.
2. *Cascading Style Sheets* (CSS) is supported to control design of every component without the need to intervene code of the application. Because every JavaFX attribute starts with a prefix `-fx-` it is easily distinguishable from different components and other parsers than JavaFX will ignore them.
3. Extensive amount of libraries for plotting graphs, displaying data. Mostly well documented and rich API [16].

We also chose the JavaFX platform because of the author familiarity with it out of all platforms that are used for creating graphical applications. The features that were just mentioned and the fact that the logic of the application is coded in Java language, were critical for making this choice. We can inspect the final project structure in the following Figure 2.4.

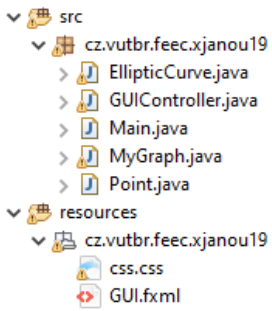

Fig. 2.4: Final structure of the JavaFX project in Eclipse environment

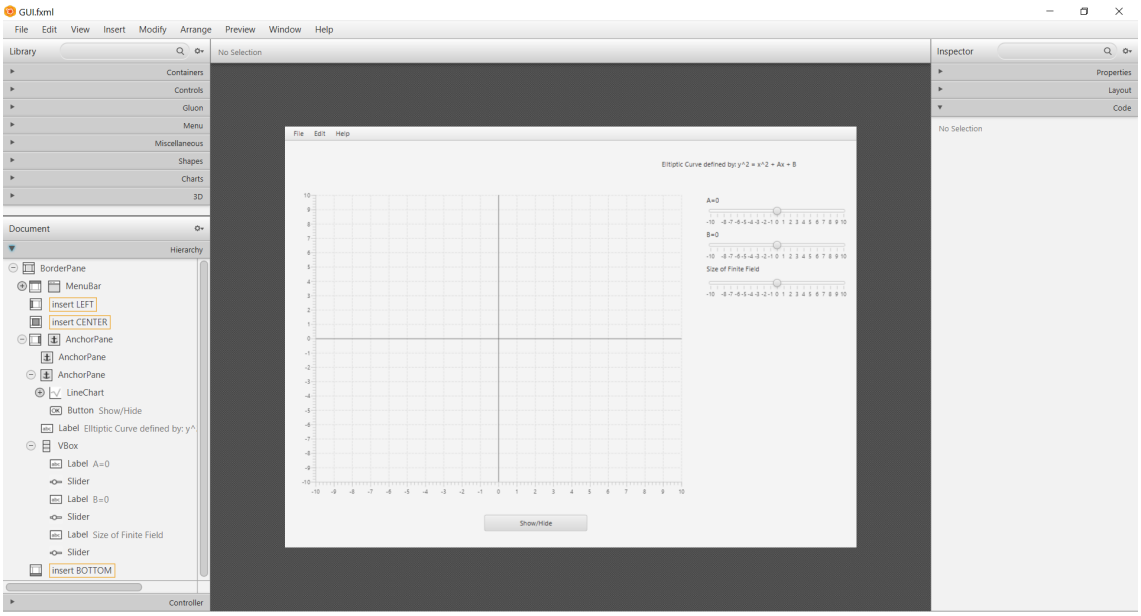

Fig. 2.5: Designing the GUI in SceneBuilder

The description of each class is given in Table 2.1.

Tab. 2.1: Overview of created classes with description

CLASS	DESCRIPTION
`EllipticCurve`	Class, where object-oriented representation of an elliptic curve in Short-Weierstrass form is defined. Contains necessary properties for domain parameters, List of points and respective methods for approaching properties (getters and setters), and methods for providing all proposed features.
`GUIController`	Class which controls graphical elements created in `GUI.fxml`. It is directly linked with it by anotating the graphical elements with `@FXML` anotation.
`Main`	The Main Class which launches the application. Its main job is to load the `GUI.fxml` class as a root of the Scene.
`MyGraph`	In this class the author implemented all functions for plotting the graphical representation of elliptic curve. It uses the JavaFX element `XYChart`[4] for displaying the graph. He created functions `plotPoint` and `plotLine` which are used to plot the graph of the curve only in given range.
`Point`	Class contains object-oriented representation of point on elliptic curve with respective properties and methods, also implements methods for comfortable display of point properties and methods for converting projective coordinates into affine system, while saving all information given by projective representation.
`css`	A CSS class where the style of certain elements is specified.
`GUI.fxml`	In this class all the graphical elements are defined. The code can be edited manually or as in our case, generated by Scene Builder. It is necessary to specify which Controller will be used in order for the application to work. In our case we specified the `GUIController` to be our Controller class.

We use commands described in our SageMath section 2.1 which we pass as a String variable to method `interactWithSage()` to communicate with SageMath within the client application. In the following subsections, we will analyze the created JavaFX application from the users' point of view. Four tabs were created, each offering different functionality. Sample screenshots were made to support the explanation given in the following subsections and can be seen in Appendix A.

2.3.1 Curve Graph in Real numbers Tab and the Control panel

The first tab shows the graph of elliptic curve in $\mathbb{R}$ which dynamically changes based on the values a, b given by the user. The user can change the values interacting with the 2 sliders in the Control panel, which is located by the right side of the application as can be seen in Figure A.1. The user also defines the characteristics of prime field $\mathbb{F}_p$. The available values a, b are from -10 to 10, which should suffice any school example and the benefit of using sliders is that the programmer does not have to deal with invalid inputs, as the only available values are integers. Because $\mathbb{F}_p$ is a prime field, the only available values for its characteristics are prime numbers ranging from 5 to 97. Prime characteristics 2 and 3 are excluded by definition of the Short Weierstrass form, given in subsection 1.2.1 of the Background chapter. The user can change the characteristics interacting with the ComboBox.

After clicking a button labeled "Get Curve Details", the SageMath server will be called to process the user request and will in result print order of curve ($\#E$), add possible orders of points and populate the Table of points with point coordinates and their respective orders.

There are situations, in which the combination of parameters given by the user would result in defining a singular curve, in other words, the curve does not satisfy equation 1.1. Trying to define such a curve in SageMath would cause an exception, therefore we do not allow it in our program. If the user manages to define such a curve, button "Get Curve Details" will change its mode to disabled and the user receives a notification about this event. This situation is demonstrated in the following Figure 2.6.

We created two methods which are in charge of controlling this event. First method, noted `isSingular()` (for inspection, check Appendix B.1), checks if a curve is singular and based on the result returns boolean - true or false. This method is called within our second method, `checkCorrectInput()` (source code in Appendix B.2). The second method based on the result of our first method either displays a warning and disables the "Get Curve details" button, or clears the warning and enables the button.

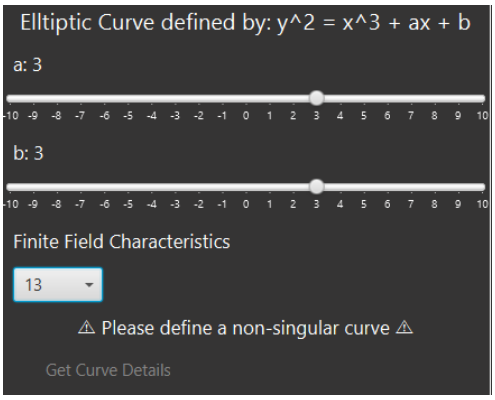

Fig. 2.6: Notification about singular elliptic curve definition

2.3.2 Point addition Tab

In the next tab, the user can perform addition of 2 points on elliptic curve, provided he has defined curve parameters before and has clicked the button "Get Curve Details". To perform point addition, all the user has to do, is select 2 points from the table view "Table of points" by clicking on them. Once he has done that, the button with the equal sign will become available and simply by clicking it, he will get the resulting point.

Because there are different cases of point addition, as we pointed out in section 1.1.2, we added a feature, which informs the user about what kind of addition he has performed. After the addition is performed, the application will display a picture, which illustrates different situations of point addition, depending on which points on the curve he has chosen to add together. The pictures shown to the user fully correspond with figures describing different cases of point addition in section 1.1.2. For example, we can see that in Figure A.2 the user has performed point addition, which corresponds with the case described in paragraph 1 and is illustrated on Figure 1.1.

To implement this feature, we created a method in our JavaFX application called `analyzePointAddition()`. This method can be inspected in Appendix B.3 of this thesis.

2.3.3 Scalar point multiplication Tab

The Point Multiplication tab (Figure A.3) offers the user to multiply the chosen point by a positive integer. First, the user chooses a point from our table view "Table of points" which unlocks the possibility of choosing the value from the ComboBox. The available values are 1 to n, where n is the order of the point. It is possible to use any integer to multiply a point, however, we decided to give only these numbers, to

educate the user about the fact that is best illustrated by equation 1.5. In other words, we give the user scalars that generates a full set of results, before it enters a new cycle. A short explanation of this fact is also provided within the window of our application to better inform the user.

2.3.4 Elliptic Curve Diffie-Hellman Tab

The fourth and perhaps most complex tab allows the user the simulate Diffie-Hellman key exchange as described in section 1.4.2. Supposing the user has already defined parameters of the elliptic curve and characteristics of $\mathbb{F}_P$, he can then proceed to choose the root point, which is the generator used as a starting point for both communicating parties Alice and Bob. After the user chooses root point, a table containing the domain parameters is displayed as seen in Figure 2.7.

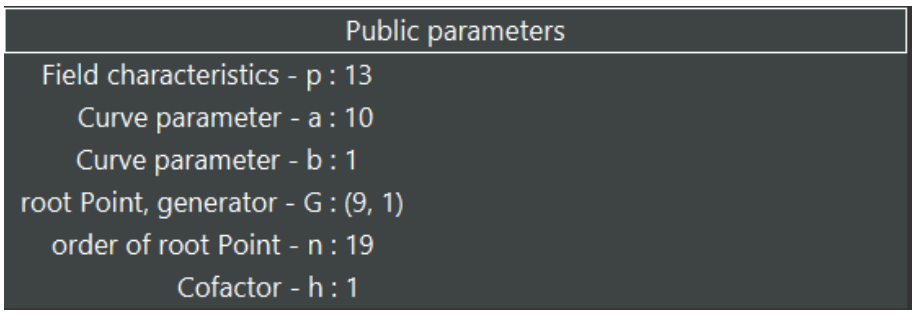

Fig. 2.7: Domain parameters displayed in JavaFX application

Our user represents both of the communicating parties, therefore he must choose secret values α and β for both, Alice and Bob in order to proceed. After choosing that, the private values are loaded onto the screen and are displayed in the equation of points A, B, K. At the same time the button labeled "Compute" is unlocked, enabling the user to execute the key exchange. As a result, a secret point K is displayed on both sides and is also highlighted at the bottom of the window. A successful key exchange is shown in Figure 2.8, while the full window is available for inspection in Appendix A.4.

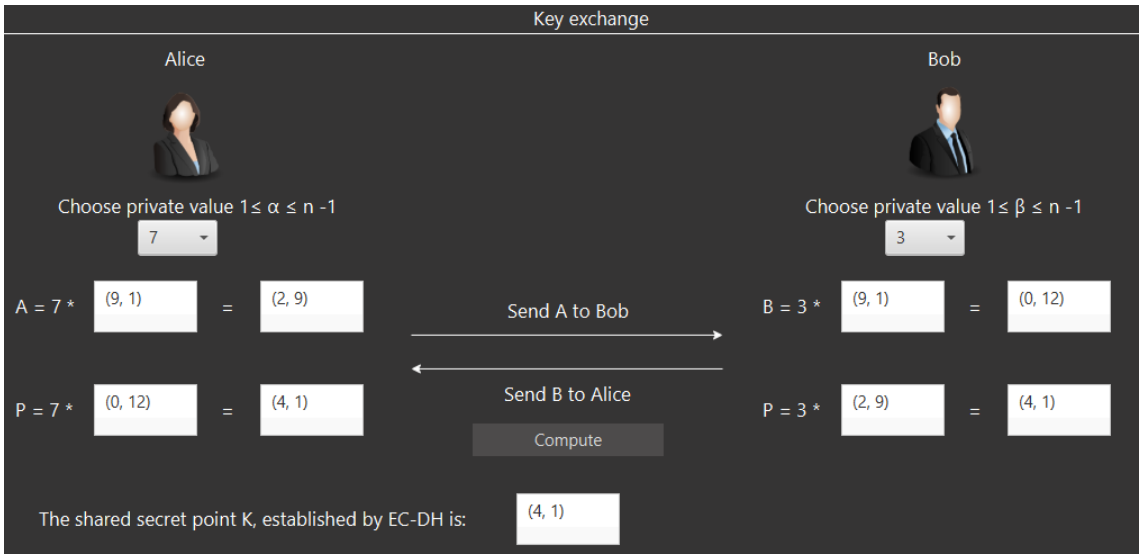

Fig. 2.8: Successful establishment of secret point K using ECDH protocol

2.4 Result discussion

In this section, we will try to shortly discuss the results achieved namely in the implementation part of this thesis and mention few caveats of this particular solution.

SageMath suitability for our project

SageMath works as the mathematical engine in our proposed solution. However, it is not without caveats in our particular solution. One is the startup time of our SageMath Server, which in testing, took about 20 seconds on the first launch on a modern desktop computer (6 core, 12 thread CPU; 16GB of DDR4 RAM; M.2 SSD storage) improving to approximately 5 seconds after several consecutive startups. However, the time raised to 2–3 minutes on an older laptop with considerably worse hardware equipment therefore users with older or not so well equipped machines can experience unusually long start of SageMath. The user has to wait every time he launches the application, to unlock the full potential of implemented JavaFX application.

On the other hand, the speed of computations and exchange speed of messages between client and server tested on a modern desktop computer was exceptional for small numbers of fields (e.q 19, 23, 29), rising to about 2–3 seconds for the maximum field number 97. This fact was one of the reasons the author decided not to raise the numbers of available parameters any further, as it would cause (due to the complexity of computations) extension of the wait time for results.

JavaFX application

Using JavaFX as a platform for the development of a graphical application had numerous of advantages, e.g. its cross-platform nature of Java language, meaning our application can technically run on any device equipped with Java virtual machine (JVM).

It provided all the necessary tools that were needed to accomplish the assignment of this thesis. However, the author is aware of minor issues regarding resizing and has decided to rather ship the application in a non-resizable format. This should impact only minority of users with small screens and resolutions going below 1920×1080 pixels.

Conclusion

In this thesis, we studied elliptic curves and their mathematical properties which are important for ECC. We presented the definition of elliptic curve and their group structure when defined over a finite field. We described 2 most used fields which are used in ECC.

Following the theory, we created project structure that would fulfill the task of this thesis. We studied SageMath and explored its functionalities related to the elliptic curves, namely defining the curve, getting the order of points and curve, getting the point of maximum order, and performing point addition.

We chose the platform for the implementation of our graphical application – JavaFX. Extensive effort was made to connect the client graphical application to the Server represented by SageMath, in order for this project to fully function. This was accomplished and afterward, the full focus was on implementing the requirements of the assignment.

For each feature proposed, the author created a tab within the JavaFX application to improve navigation through the program. As a result, the application allows the user to choose parameters of an elliptic curve in Short Weierstrass form, define characteristics of $\mathbb{F}_p$ and based on that show graph of elliptic curve in real numbers, table of points with their respective orders, the order of the curve, and possible orders of points. Moreover, the user can perform point addition, while being informed about what kind of addition he performed, perform scalar multiplication, and establish a secret key using the Elliptic Curve Diffie-Hellman protocol.

All of the required goals were accomplished and the project was implemented in a way so that is possible to work on further development.

Bibliography

[1] WASHINGTON, Lawrence C. *Elliptic curves: number theory and cryptography*. 2nd ed. Boca Raton, FL: CRC Press, 2008. ISBN 978-1-4200-7146-7.

[2] HANKERSON, Darrel R., Scott A. VANSTONE a A. J. MENEZES. *Guide to elliptic curve cryptography*. New York: Springer, 2003. ISBN 03-879-5273-X.

[3] BARKER, Elaine. *Recommendation for key management part 1: General*, Revision 4. *NIST Special Publication Part 1*, 800-57, 2016. Available from: `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf`

[4] LOZANO-ROBLEDO, Alvaro. *Number theory and geometry: an introduction to arithmetic geometry*. Providence, Rhode Island: American Mathematical Society, [2019]. ISBN 978-1-4704-5016-8.

[5] MENEZES, A., OKAMOTO, T., VANSTONE, S. *Reducing elliptic curve logarithms to logarithms in a finite field*, IEEE Transactions on Information Theory 39 (1993), 1639-1646.

[6] DZURENDA, Petr, Sara RICCI a Lukáš MALINA. *Performance Analysis and Comparison of Different Elliptic Curves on Smart Cards* [online]. , 2-10 [cit. 2019-12-02]. DOI: 10.1109/PST.2017.00050. Available from: `https://www.researchgate.net/publication/324780353_Performance_Analysis_and_Comparison_of_Different_Elliptic_Curves_on_Smart_Cards`

[7] DIFFIE, W., HELLMAN, M. *New directions in cryptography*. IEEE transactions on Information theory. 1976 Nov;22(6): 644-54.

[8] KOBLITZ, N. *Elliptic curve cryptosystems*. Mathematics of computation. 1987;48(177): 203-9.

[9] DZURENDA, Petr. *Cryptographic protection of digital identity*. Brno, 2019, 159 p. Doctoral thesis. Brno Unversity of Technology, Faculty of Electrical Engineering and Communication, Department of Telecommunications.

[10] SILVERMAN J.H., SUZUKI J. *Elliptic Curve Discrete Logarithms and the Index Calculus*. In: Ohta K., Pei D. (eds) Advances in Cryptology — ASIACRYPT'98. ASIACRYPT 1998. Lecture Notes in Computer Science, vol 1514. Springer, Berlin, Heidelberg

[11] COHEN, Henri, Gerhard FREY a Roberto AVANZI. *Handbook of elliptic and hyperelliptic curve cryptography*. Boca Raton: CRC Press, 2006. ISBN 15-848-8518-1.

[12] BERNSTEIN, Daniel J. a Tanja LANGE. Montgomery curves and the Montgomery ladder. *IACR Cryptology ePrint Archive* [online]. **2017**(2017), 1-293 [cit. 2019-12-16]. Available from: `https://eprint.iacr.org/2017/293`

[13] *The Sage Reference Manual - SageMath Documentation* [online]. The Sage Development Team, 2019 [cit. 2019-12-17]. Available from `https://doc.sagemath.org/html/en/reference`

[14] The Sage Reference Manual - SageMath Documentation. *Elliptic Curves* [online]. The Sage Development Team, 2019 [cit. 2019-12-17]. Available from `http://doc.sagemath.org/html/en/constructions/elliptic_curves.html`

[15] *Java Platform, Standard Edition (Java SE) 8, JavaFX: Getting Started with JavaFX*, [online]. available from: `https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#A1141718`

[16] *JavaFX8, API overview-summary*,[online]. available from: `https://docs.oracle.com/javase/8/javafx/api/overview-summary.html`

[17] *Java™ Platform, Standard Edition 7 API Specification, Class ProcessBuilder*,[online]. available from: `https://docs.oracle.com/javase/7/docs/api/java/lang/ProcessBuilder.html`

[18] P. Gallagher, *Digital signature standard (dss),* Federal Information Processing Standards Publications, volume FIPS 186-4, pp. 1-121, 2013.

[19] D. R. L. Brown, "Sec 2: Recommended elliptic curve domain parameters, version 2.0," *Standards for Efficient Cryptography*, pp. 1-33, 2010.

[20] "American national standards institute," "public key cryptography for the financial service industry: The elliptic curve digital signature algorithm (ecdsa)", ANSI X9.62, pp. 1-55, 2005.

[21] B Black. J. W. Bos, C. Costello, P. Longa and M. Naehrig, *Elliptic curve cryptography (ecc) nothing up my sleeve (nums) and curve generation*, Tech. Rep., 2014.

[22] M. Lochter and J. Merkle, *Elliptic curve cryptography (ecc) brainpool standard curves and curve generation,* Tech. Rep., 2010.

[23] P. S. Barreto and M. Naehrig, *Pairing-friendly elliptic curves of prime order*, International Workshop on Selected Areas in Cryptography. Springer, 2005, pp. 319-331.

[24] H. Edwards, *A normal form for elliptic curves*, Bulletin of the American Mathematical Society, vol. 44, no. 3, pp. 393-422, 2007.

[25] *Mintty - Cygwin Terminal emulator* [online]. [cit. 2020-06-06]. Available from: `https://mintty.github.io/`

[26] Java ProcessBuilder examples. *Mkyong* [online]. 2019 [cit. 2020-06-06]. Available from: `https://mkyong.com/java/java-processbuilder-examples/`

[27] *Socket (Java Platform SE 7)* [online]. Oracle [cit. 2020-06-06]. Available from: `https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html`

[28] *BufferedReader (Java Platform SE 8)*. Oracle [cit. 2020-06-06]. Available from: `https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html`

[29] *Running Sage from other languages with high(er) performance?*. [online]. By Woodgnome, licended under CC BY-SA 3.0. Available from: `https://ask.sagemath.org/question/23431/running-sage-from-other-languages-with-higher-performance/`

List of symbols, physical constants and abbreviations

$\mathbb{R}$	set of real numbers
$\mathbb{C}$	set of complex numbers
$\mathbb{Q}$	set of rational numbers
$\mathbb{F}_q$	finite field of q elements
DSA	Digital signature algorithm
RSA	Rivest-Shamir-Adleman
DH	Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
ECDH	Elliptic Curve Diffie-Hellman
ECC	Elliptic curve cryptography
GF	Galois Field
GUI	graphical user interface
API	Application programming interface
IDE	Integrated development environment
CSS	Cascading Style Sheets
NIST	National Institute of Standards and Technology
DLP	Discrete logarithm problem
ECDLP	Elliptic curve discrete logarithm problem
IF	Integer factorization
SSH	Secure Shell
TLS	Transport Layer Security
IKE	Internet Key Exchange
IPsec	Internet Protocol Security
VPN	Virtual Private Network
MITM	Man In the Middle
AES	Advanced encryption standard
CAS	Computer algebra system

List of appendices

A Captures of application tabs and user interface

A.1 Curve Graph in Real numbers and the Control panel capture

A.2 Point addition tab capture

A.3 Scalar point multiplication tab capture

A.4 Elliptic curve Diffie-Hellman tab capture

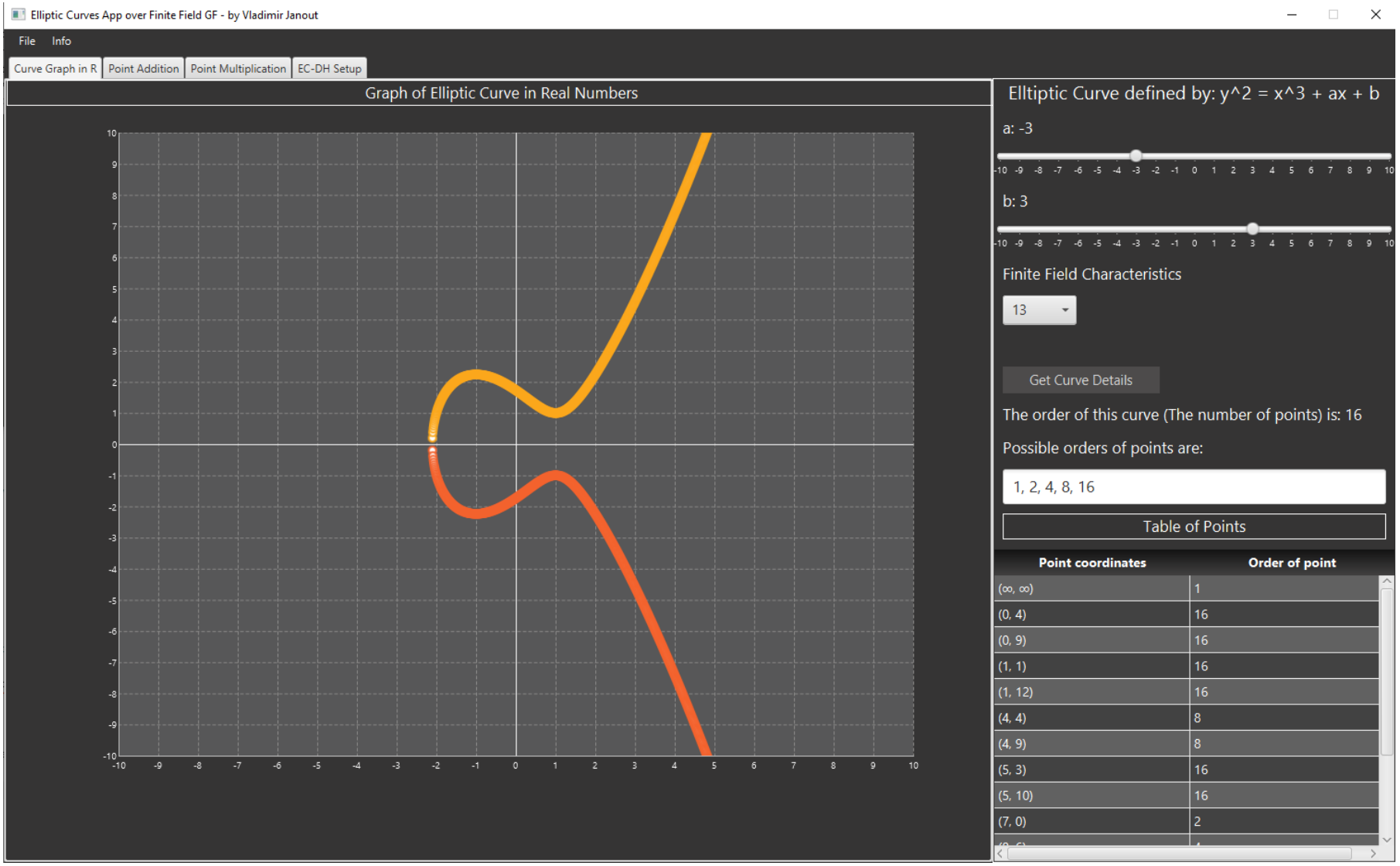

Fig. A.1: The tab showing elliptic curve graph in $\mathbb{R}$ and the Control panel

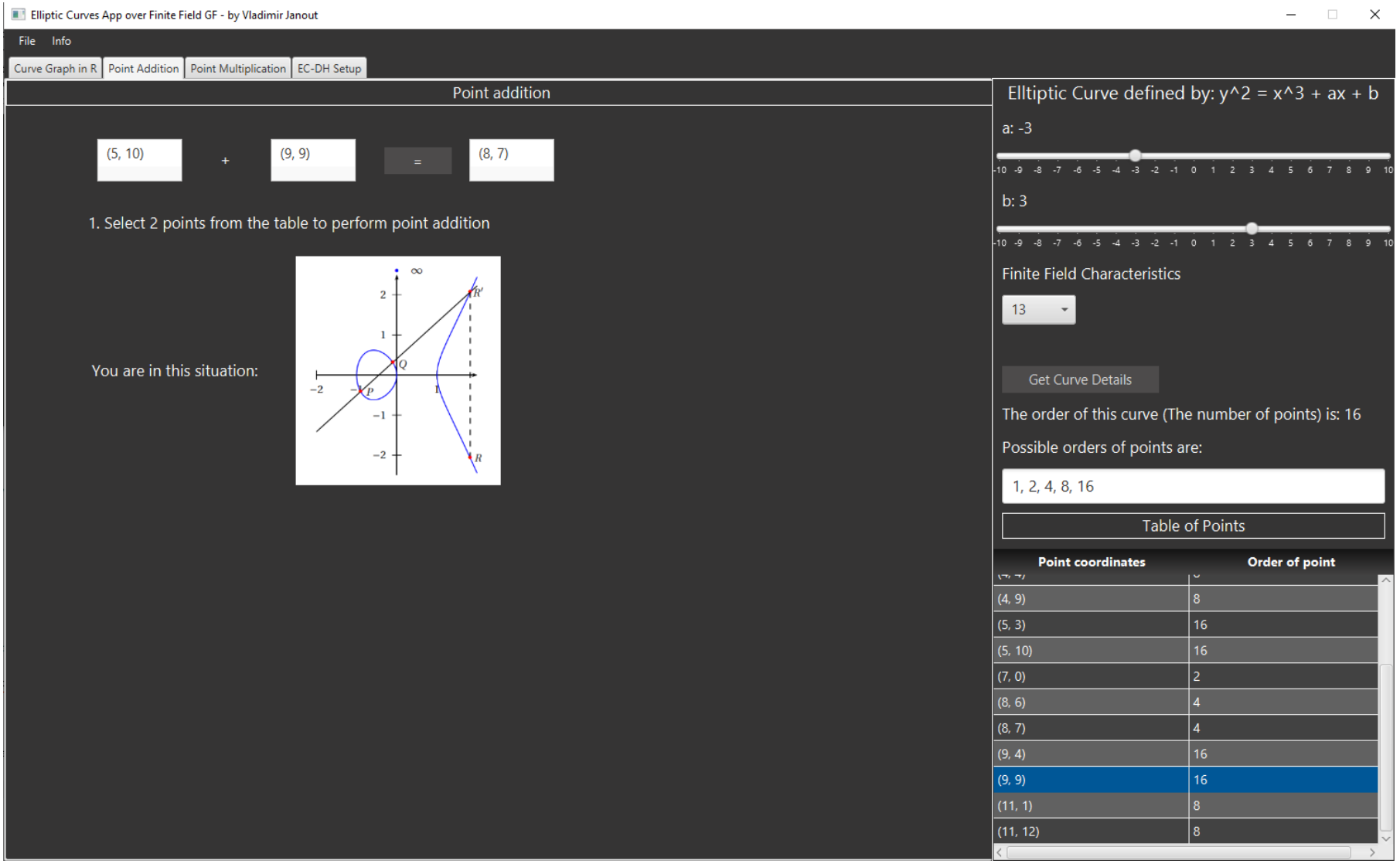

Fig. A.2: Point addition tab showing addition of 2 points and the curve with respective picture for different situations

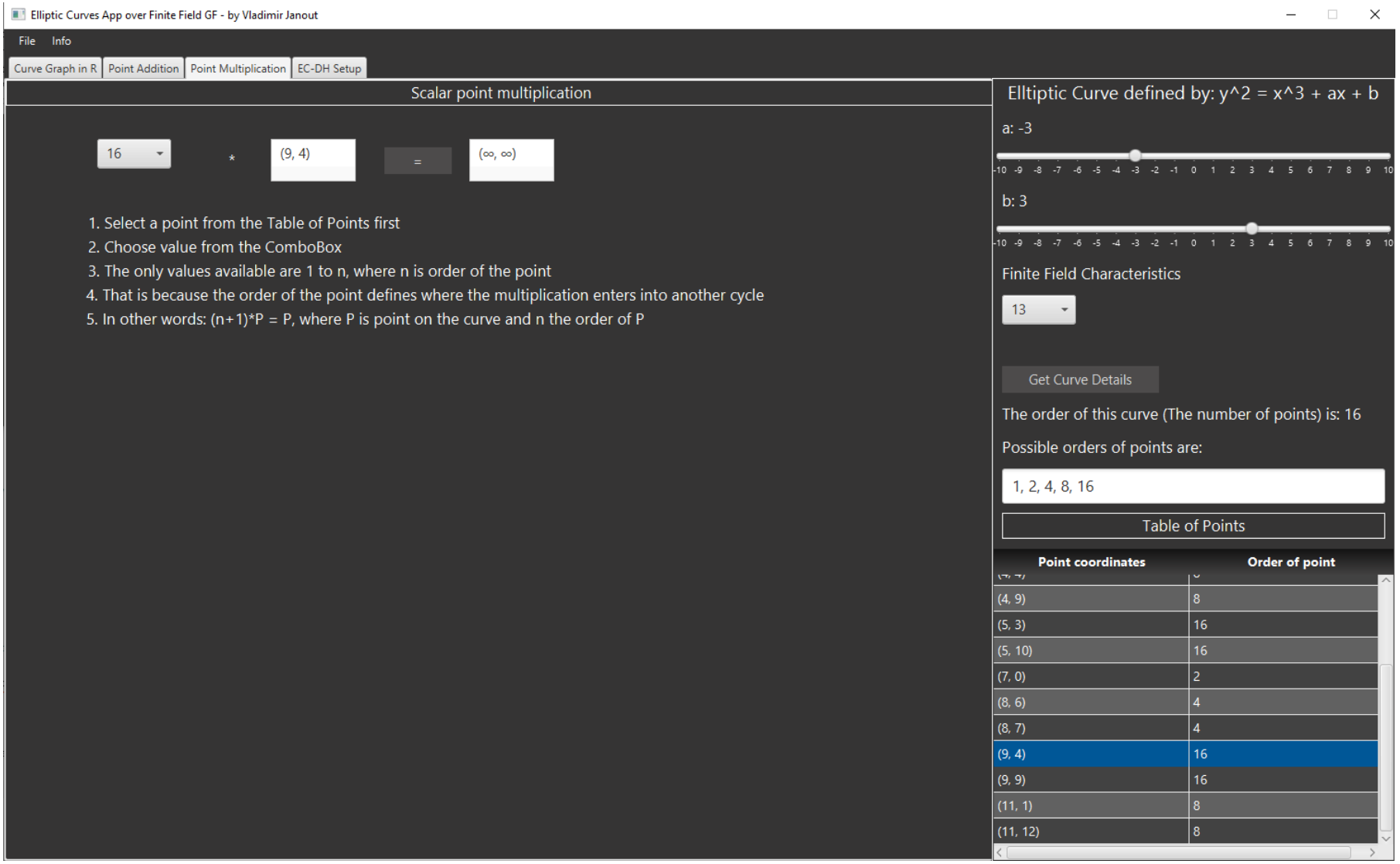

Fig. A.3: Scalar point multiplication tab with short manual and explanation

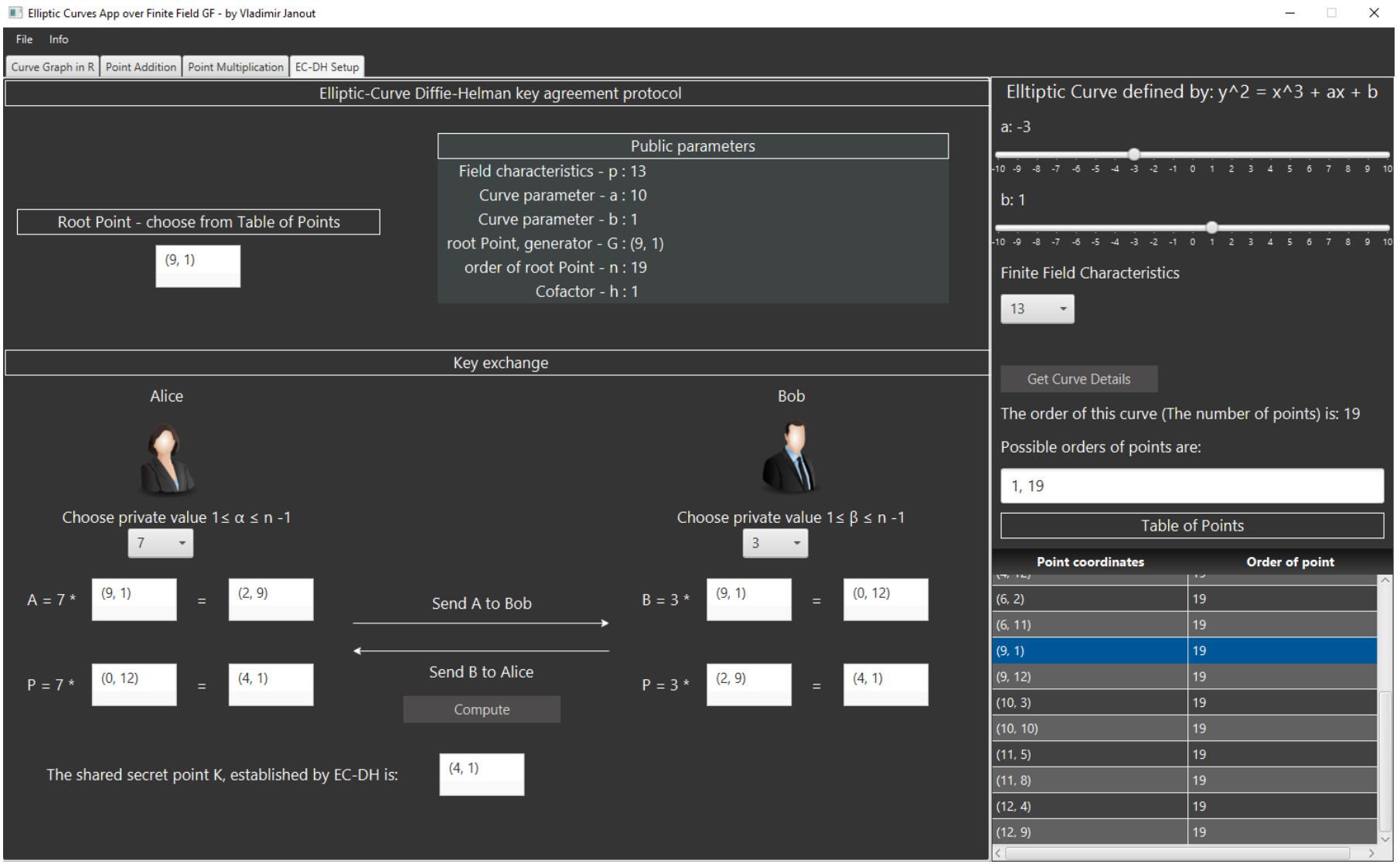

Fig. A.4: Elliptic curve Diffie-Hellman tab displaying whole processes of key exchange between the two parties, Alice and Bob resulting in agreed upon point K

B Source code of selected implemented methods

B.1 Method for checking curve singularity

```
private boolean isSingular() {  //This block of code
   is reducing a and b modulus field if they are
   reducible
  if(field <= Math.abs(a)) {
    a = a%field;
  }
  //if a and b are -value of field - add them to be
     possitive
  if(a < 0) a = a + field;

  if(field <= Math.abs(b)) {
    b = b%field;
  }
  if(b < 0) b = b + field;
  //curve is singular if 4a^3+27b^2 = 0 (mod field)
  if((4*(a*a*a) + 27*(b*b))%field == 0) return true;
  else return false;
}
```

B.2 Method for singular curve notification

```
private void checkCorrectInput() {
  if(isSingular()) {   //u26a0 is unicode for
     displaying warning sign
    getButton.setDisable(true);
    labelWarning.setText("\u26a0 Please define a non-
       singular curve \u26a0");
    return;
  }
  getButton.setDisable(false);
  labelWarning.setText("");
}
```

B.3 Method for displaying point addition cases

```
    private void analyzePointAddition() {
  //when trying to add the point in infinity to real
     point - the result will be real point
  if((p.isInfinity() && !q.isInfinity()) || !p.
     isInfinity() && q.isInfinity()) {
    situationImage.setImage(i5);
    return;
  } //doubling a point with y coordinate equal to 0
     will result in the point in infinity
  if(p.getX() == q.getX() && (p.getY() == 0 && q.getY()
      == 0)) {
    situationImage.setImage(i1);
    return;
  } //this should occur when the user performs
     doubleing 2P - with the Y_p != 0, P is real point
  if(p.getX() == q.getX() && p.getY() == q.getY() && p.
     getY() != 0 && (!p.isInfinity() || !q.isInfinity()
     )) {
    situationImage.setImage(i3);
    return;
  } //two real points with different coordinates -
     regular P+Q = R
  if(p.getX() != q.getX() && p.getY() != q.getY() && (!
     p.isInfinity() && !q.isInfinity())) {
    situationImage.setImage(i4);
    return;
  } //adding inverse points will result in the point
     in infinity
  if(p.getX() == q.getX() && p.getY() != q.getY()) {
    situationImage.setImage(i2);
    return;
  } //infinity + infinity = you guessed it - infinity
  if(p.isInfinity() && q.isInfinity()) {
    situationImage.setImage(i6);
    return;
  }
}
```

C Installation and launch Manual

SageMath Installation and setup

It is necessary to first download **SageMath** in version 8.9 for Windows, available from the official SageMath webpage: `https://www.sagemath.org/`, direct download can be started using this link. The version is important because newer versions of SageMath are running Python 3 instead of 2 which can cause issues with our solution.

After accepting the license agreement it is important to remember the **Sage home directory** as we will copy our python script there. It is also convenient to select: "create desktop and start menu icons".

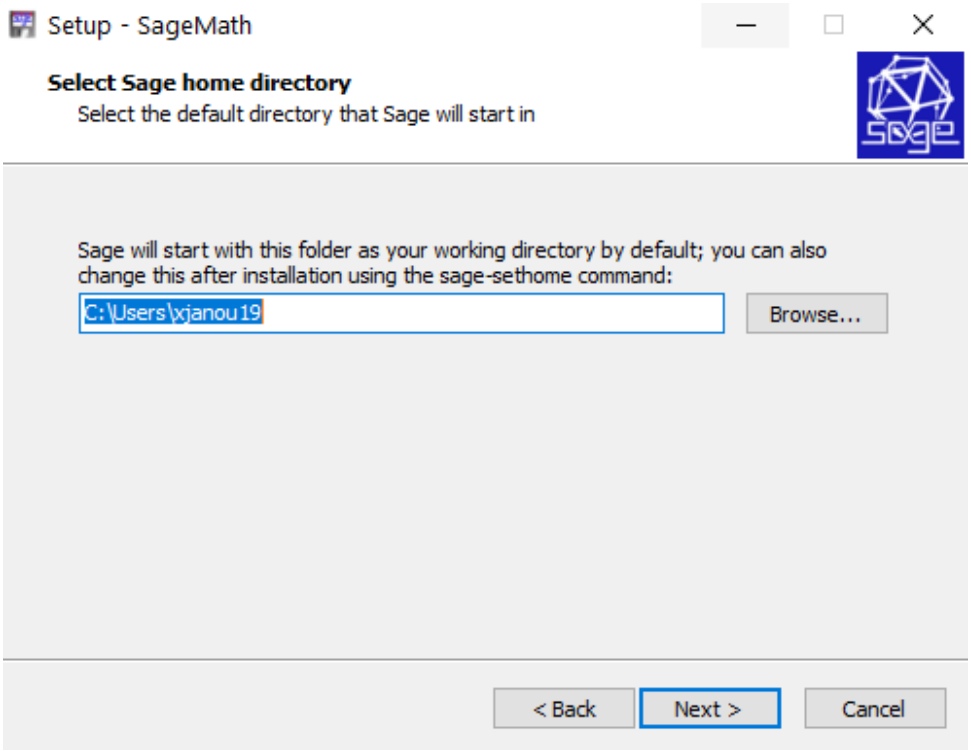

Fig. C.1: Choosing the Sage home directory

After installing SageMath, copy `sage-daemon.py` file to the **Sage home directory**.

JavaFX application installation

Using the `Application for Elliptic Curve Cryptography-1.0.exe` installer will install the application together with all necessary libraries, dependencies and JVM to location: *C:\Users\<Username>\AppData\Local\Application for Elliptic Curve Cryptography*. In this folder we can find executable file: `Application for Elliptic Curve Cryptography.exe` which will launch the JavaFX application.

Launch of the solution

After installing **SageMath 8.9**, **JavaFX application** and coppying `sage-daemon.py` file into the **Sage Home Directory** just launch the JavaFX application using `Application for Elliptic Curve Cryptography.exe` in the installation directory. The **SageMath Server** should be started automatically. If that is not the case, continue reading next subsection. It is best to close the app using the JavaFX application by clicking on **`File`** $\rightarrow$ **`Close`** from the Menu bar.

Manual launch of SageMath Server

In case the SageMath server did not start automatically, here we provide short description how to launch it manually.

Open the **SageMath 8.9 Shell** and launch the SageMath Server using following command and wait for it to launch:

```
sage -python sage-daemon.py
```

Successful launch of the SageMath Server should look like in Figure C.2

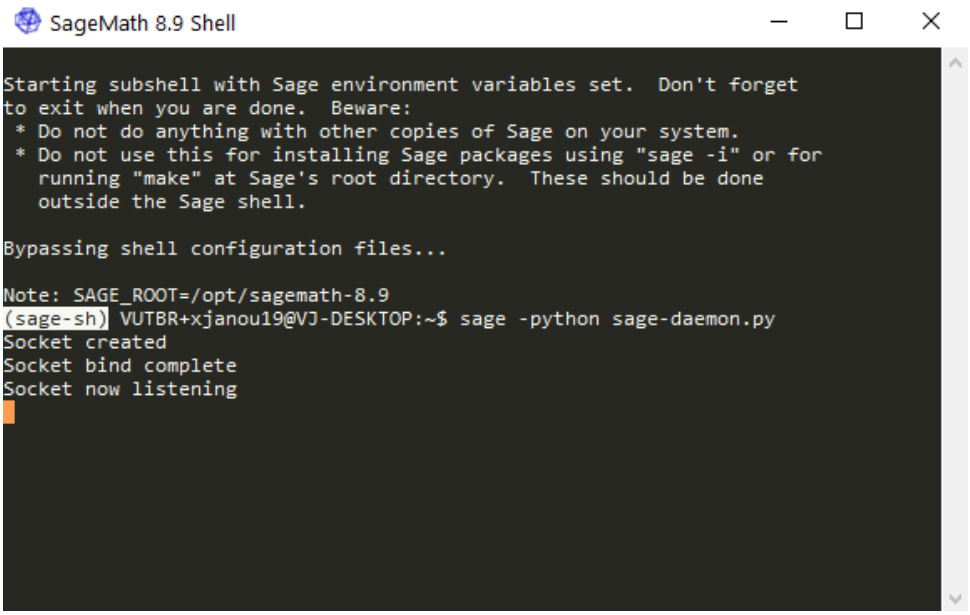

Fig. C.2: Successful launch of the SageMath Server

After that, launch the JavaFX application using `Application for Elliptic Curve Cryptography.exe` and start using the application.

D Contents of enclosed data storage

- `JavaFX_application_installer` ..... folder with JavaFX application installer
  - `Application for Elliptic Curve Cryptography-1.0.exe`
- `JavaFX_source` ..... JavaFX application source code folder
- `Script` ..... folder containing SageMath Server script
  - `sage-daemon.py`
- `xjanou19_bp.pdf` ..... text version of bachelor thesis in PDF
- `README.txt` ..... installation and launch instruction manual